Confusion in Code Reviews: Reasons, Impacts, and Coping Strategies

Felipe Ebert*, Fernando Castor*, Nicole Novielli†, Alexander Serebrenik‡
*Federal University of Pernambuco, Brazil, {fe,castor}@cin.ufpe.br
†University of Bari, Italy, nicole.novielli@uniba.it
‡Eindhoven University of Technology, The Netherlands, a.serebrenik@tue.nl

***Abstract*—Code review is a software quality assurance practice widely employed in both open source and commercial software projects to detect defects, transfer knowledge and encourage adherence to coding standards. Notwithstanding, code reviews can also delay the incorporation of a code change into a code base, thus slowing down the overall development process. Part of this delay is often a consequence of reviewers not understanding, becoming confused by, or being uncertain about the intention, behavior, or effect of a code change.**

We investigate the reasons and impacts of confusion in code reviews, as well as the strategies developers adopt to cope with confusion. We employ a concurrent triangulation strategy to combine the analyses of survey responses and of the code review comments, and build a comprehensive confusion framework structured along the dimensions of the review process, the artifact being reviewed, the developers themselves and the relation between the developer and the artifact. The most frequent reasons for confusion are the *missing rationale*, *discussion of non-functional* requirements of the solution, and *lack of familiarity with existing code*. Developers report that confusion *delays* the merge decision, *decreases review quality*, and results in *additional discussions*. To cope with confusion developers *request information*, *improve familiarity with existing code*, and *discuss off-line*.

Based on the results, we provide a series of implications for tool builders, as well as insights and suggestions for researchers. The results of our work offer empirical justification for the need to improve code review tools to support developers facing confusion.

***Index Terms*—code review; confusion; survey; cards sorting.**

I. Introduction

Code review is a an important practice for software quality assurance, which has been widely adopted in both open source and commercial software projects [1], [2], [3], [4], [5]. The benefits of code reviews are well-known. Active participation of developers in code reviews decreases the number of post-release defects and improves the software quality [6], [7]; knowledge transfer and adherence to the project coding standards are additional benefits of code reviews [8], [9], [10].

However, code reviews also incur cost on software development projects as they can delay the merge of a code change in the repository and, hence, slowdown the overall development process [11], [12]. Indeed, the time spent by a developer reviewing code is non-negligible [1] and may take up to 10-15% of the overall time spent on software development activities [13], [6]. The merge of a code change in the repository can be even further delayed if the reviewers experience difficulties in understanding the change, *i.e.*, they are not certain about its correctness, run-time behavior and impact on the system [6], [9], [14], [15], [16]. As such, we believe that a proper understanding of the main reasons for confusion in code reviews represents a necessary starting point towards reducing the cost and enhancing the effectiveness of this practice, thus improving the overall development process.

We focus on confusion experienced by developers during code review, its reasons and impacts, as well as on the strategies adopted by developers to cope with it. By *confusion* we mean "*any situation where the person is uncertain about anything or unable to understand something*" [17]. We do not distinguish between lack of knowledge, confusion, and uncertainty. Indeed, confusion (which also encompasses doubt and uncertainty) and lack of knowledge are strictly connected (*e.g.*, confusion could be determined by lack of knowledge) [18].

Our goals are threefold. First, we aim at obtaining empirically-driven actionable insights for both researchers and tool builders, on what are the main causes of confusion in code reviews. Thus, we formulate our first research question:

RQ1.*What are the reasons for confusion in code reviews?*

We have observed that the three most frequent reasons for confusion are *missing rationale*, *discussion of the solution: non-functional*, and *lack of familiarity with existing code*.

Second, while confusion can be expected to negatively affect code reviews, we would like to identify specific impacts of confusion. By monitoring these impacts developers and managers can curb the undesirable consequences. As such, we formulate our second research questions:

RQ2. *What are the impacts of confusion in code reviews?*

Our results suggest that the merge decision is *delayed* when developers experience confusion, there is an increase in the number of messages exchanged during the discussion, and the *review quality decreases*. However, we also observed unexpected consequences of confusion, such as helping to find a *better solution*. This suggests that communicating uncertainty and doubts might be beneficial for collaborative code development, *i.e.*, by inducing critical reflection [19] or triggering knowledge transfer [9].

Finally, we believe that understanding the strategies adopted by the developers to deal with confusion can further inform the design of tools to support code reviewers in fulfilling their information needs associated to the experience of uncertainty and doubt. As such, we formulate our third research question:

RQ3. *How do developers cope with confusion during code reviews?*

978-1-7281-0591-8/19/$31.00 © 2019 IEEE

The results suggest that developers try to deal with confusion by *requesting information*, *improving the familiarity with existing code*, and *discussing off-line* the code review tool. We also found that confusion might simply induce developers to *blindly approve* the code change, regardless its correctness.

The main contribution of our work is a **comprehensive framework for confusion in code reviews** including reasons, impacts, and coping strategies. To address our research questions, we implement a concurrent triangulation strategy [20] by combining a survey ('what people think') with analysis of the code review comments ('what people do') from the dataset provided by Ebert *et al.* [17]. The data collected and manually annotated during the study are released to enable follow-up studies.[1] Based on the analysis of the framework we formulate a series of suggestions for tool builders and researchers.

The findings of our study complement recent research on comprehension in code reviews, *i.e.*, the study by Ebert *et al.* [17] proposing a model to identify confusion in code reviews and the one by Pascarella *et al.* [11], focusing on understanding the information needs of code reviewers.

The remainder of the paper is organized as follows. Section II presents the necessary background information. Section III describes the methodology, and Section IV—results. We discuss the results and their implications in Section V. The threats to validity are discussed in Section VI. Section VII presents the related work and Section VIII concludes.

II. Code Review

Formal code review was first defined by Fagan in 1976 as a software inspection practice, a structured process for reviewing source code with the single goal of finding defects, usually conducted by groups of reviewers in extended meetings [21]. Open source projects have been employing some form of code reviews for more than two decades. In particular, this practice became very popular with the Linux operating system kernel [22]. Based on the reviewing practices observed at Microsoft in 2013, Bacchelli and Bird [9] have defined the concept of *modern code review*, a frequent and informal process supported by *ad hoc* tools, *i.e.*, a more *lightweight* process compared to the one depicted by Fagan's definition. From here on we refer to modern code review as *code review*.

Depending on when the review is done, one can distinguish between *review-then-commit (pre-commit)*, which involves reviewing the code before it is integrated into the main repository, and *commit-then-review (post-commit)*, which involves reviewing the code after its integration in the main repository [23]. Figure 1 presents an overview of the typical code review process following the review-then-commit method. The code review is triggered by the author submitting a code change (1). The reviewers then check and verify the change (2) based on its correctness, adherence to the project guidelines, conventions, and quality criteria. If the code change does not satisfy these requirements, the reviewers either ask the author to revise it or to submit a new one (3). Once the reviewers are satisfied, the change is integrated in the code repository (4). Conversely, if the change does not satisfy the reviewers, it is rejected and the code review is abandoned (5).

[1]The dataset is available at https://github.com/felipeebert/confusion-code-reviews

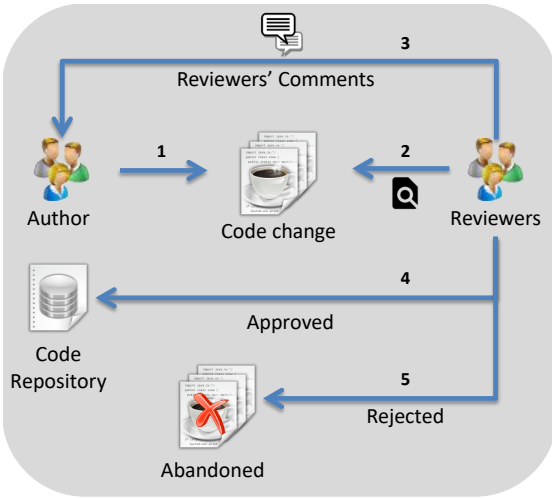

Fig. 1. Code review process.

III. Methodology

In the following, we describe how we implement the concurrent triangulation strategy [20]. First, we conduct a survey to understand "what developers say" (Section III-A). Then we analyze the code review comments to understand "what developers do" (Section III-B). Finally, we compare and contrast the findings of the two analyses (Section III-C).

A. Surveys

In literature, a theory is missing to describe what are the reasons for confusion in code reviews, the impact of confusion on the development process, and what coping strategies developers employ to deal with confusion. As such, to answer our **RQs** we opt for grounded theory building [24], [25]. We implement an iterative approach. During each iteration, we administer a survey to developers involved in code reviews. We ask developers that already answered the survey during one of the previous iterations to refrain from answering it again.

1) Survey design: The survey was designed according to the established best practices [26], [27], [28], [29]: prior to asking questions, we explain the purpose of the survey and our research goals, disclose the sponsors of our research and ensure that the information provided will be treated in a confidential way. In addition, we inform the participants about the estimated time required to complete the survey, and obtain their informed consent. The invitation message includes a personalized salutation, a description of the criteria we used for participant selection, as well as the explanation that there would not be any follow up if the respondent did not reply. This last decision also implies that we did not send reminders.

The survey starts with the definition of confusion as provided in Section I, followed by a question requiring the partic-

TABLE I
SURVEY QUESTIONS. THE QUESTIONS MARKED "*" WERE ONLY USED IN THE FIRST SURVEY, "+" —ONLY IN THE SECOND AND THIRD SURVEYS.

	Electronic Consent
0.	Please select your choice below. Selecting the "yes" option below indicates that: i) you have read and understood the above information, ii) you voluntarily agree to participate, and iii) you are at least 18 years old. If you do not wish to participate in the research study, please decline participation by selecting "No".
	Definition of Confusion
	The remainder of this survey is dedicated to "confusion". We do not make a distinction between lack of knowledge, confusion, or uncertainty. For simplicity reasons, we use the "confusion" to refer to all these terms.
1.	By clicking "next" you declare that you understand the meaning of confusion on this survey.
	Review-Then-Commit
2.+	Have you ever taken part in a "review-then-commit" type of code review (*i.e.*, the code is reviewed before it is integrated into the main repository), either in the role of author or reviewer?
	When reviewing code changes
3.	Developers might feel confused or think that they do not understand the code they review. How often did you feel this way when reviewing code changes?
4.	What usually makes you confused when you are reviewing code changes? Please explain which factors led you to be confused.
5.	Please describe a change you have been reviewing that has confused you.
6.	How does the confusion you experience as a reviewer impact code review?
7.	What do you usually do to overcome confusion in code reviews? Please explain the actions you take when you feel confused.
8.*	When you do not understand a code change, do you usually express this in general comments or in inline comments? Please explain why in the "other" field.
	When authoring code changes
9.	Developers who authored code changes might feel confused or think that they do not understand something when their code is being reviewed. How often did you feel this way when your code has been reviewed?
10.	What usually makes you confused during the code review when you are the author of the code changes? Please explain which factors led you to be confused.
11.	Please describe a change you have been authoring that has confused you.
12.	How does confusion you experience as the code change author impact the code review?
13.	What do you usually do to overcome confusion in code reviews? Please explain the actions you take when you feel confused.
14.*	When you do not understand a code change, do you usually express this in general comments or in inline comments? Please explain why in the "other" field.
	Background
15.	What is your experience as a developer?
16.	What is your experience as a code reviewer?
17.	How often do you submit code changes to be reviewed?
18.	How often do you review code changes?
19.*	Do you have the merge approval right (*i.e.*, the permission to give +2) in Gerrit at least for one software development project?
20.*	Which option would best describe yourself? - I contribute to Android voluntarily. - I'm employed by a company other than Google and I contribute to Android as part of my job. - I'm employed by Google and I contribute to Android as part of my job.- Other.
	Results
21.	Would you like to be informed about the outcome of this study and potential publications? Please leave a contact email address.
22.	Would you be willing to be interviewed afterwards?
23.	Please add additional comments below.

ipants to confirm that they understood the definition. Next, we ask two series of questions: the questions were essentially the same but were first asked from the perspective of the author of the code change, and then from the perspective of the reviewer of the change (cf. Table I). Each series starts with the Likert-scale question about the frequency of experienced confusion: *never*, *rarely*, *sometimes*, *often*, and *always*. To ensure that the respondents interpret these terms consistently we provide quantitative estimates: 0%, 25%, 50%, 75% and 100% of the time. For respondents who answered anything different from *never*, we pose four open-ended questions (to get the as rich as possible data [30]): i) what are the reasons for confusion, ii) whether they can provide an example of a practical situation where confusion occurred during a code review (**RQ1**), iii) what are the impacts of confusion (**RQ2**), and iv) how do they cope with confusion (**RQ3**). Finally, we ask the participants to provide information about their experience as developers and frequency of reviewing and authoring code changes. We ask these question at the end of the survey rather than at the beginning to reduce the *stereotype threat* [29]. Prior to deploying the survey, we discussed it with other software engineering researchers and clarified it when necessary.

2) Participants: The target population consists of developers who participated in code reviews either as a change author or as a reviewer. During the first iteration we target ANDROID developers who participated in code reviews on GERRIT: 4,645 of their email addresses provided by Ebert *et al.* [17] allow us to contact the developers by email and evaluate the response rate. In the subsequent iterations, the survey was announced on FACEBOOK and TWITTER. As the exact number of developers participating in code reviews reached cannot be known we do not report the response rate for the follow-up surveys.

3) Data analysis: To analyze the survey data, we use a card sorting approach [31]. We analyze the survey responses from the first iteration using *open card sorting* [31], *i.e.*, topics were not predefined but emerged and evolved during the sorting process. After each subsequent survey iteration, we use the results of the previous iteration to perform *closed card sorting* [31], *i.e.*, we sort the answers of each survey iteration according to the topics emerging from the previous one. If the closed card sorting succeeds, this means that the saturation has been reached and sampling more data is not likely to lead to the emergence of new topics [32], [33]. In such a case the iterations stop. If, however, during the closed card sorting additional topics emerge, another iteration is required.

To facilitate analysis of the data we use axial coding [27] to find the connections among the topics and group them into dimensions. These dimensions emerge and evolve during the final phase of the sorting process, and they represent a higher level of abstraction of the topics.

As we have multiple iterations and multiple surveys answered by different groups of respondents, *a priori* it is not clear whether the respondents can be seen as representing the same population. Indeed, it could have been the case that, *e.g.*, respondents of the second survey happened to be less inclined to experience confusion than the respondents of the third survey and the reasons of their confusions are very different. This is why we first check similarity of the groups of respondents in terms of their experience as developers and code reviewers, frequency of submitting changes to be reviewed and reviewing changes as well as frequency of experiencing confusion. If the groups of respondents are found to be similar, we can consider them as representing the same population and merge the responses. If the groups of respondents are found to be different, we treat the groups separately. To perform the similarity check we use two statistical methods: i) analysis of similarities (ANOSIM) [34], which provides a way to test statistically if there is a significant difference between two or more groups of sampling units, and ii) permutational multivariate analysis of variance using distance matrices (ADONIS) [35], [36].[2]

B. Analysis of Code Review Comments

To triangulate the survey findings for the **RQs** we perform an analysis of code review comments. As a dataset we use the one provided by Ebert *et al.* [17]. Similarly to the developers contacted during the first survey iteration, this dataset pertains to ANDROID. The code reviews of ANDROID are supported by GERRIT, which enables communication between developers during the process by using general and inline comments. The former are posted in the code review page itself, which presents the list of all general comments, while the inline comments are included directly in the source code file. The dataset of Ebert *et al.* comprises 307 code review comments manually labeled by the researchers as confusing: 156 are general and 151 are inline comments.

Similarly to the analysis of the survey data, we use card sorting to extract topics from the code review comments. We conduct an open card sorting of the general comments to account for the possibility of divergent results, *i.e.*, we did not want to use the results from the surveys because what developers do often differs from what they think they do and the emergent codes might *a priori* be different from those obtained when analyzing the survey data. To confirm the topics emergent from the general comments we then perform a closed card sorting on the inline comments.

C. Triangulating the Findings

Recall that the goal of concurrent triangulation is to corroborate the findings of the study, increasing its validity. However, following Easterbrook *et al.* [20] we expect to see some differences between 'what people say' (survey) and 'what people do' (code review comments). Hence, if the topics extracted from the surveys and code review comments disagree, we conduct a new card sorting round only on the cards associated with topics discovered on the basis of the survey but not on the basis of the code review comments, or vice versa. In order not to be influenced by the results of the previous card sorting, we perform open card sorting and exclude the researchers who participated in the previous card sorting rounds. Finally, in order to finalize the comprehensive framework for confusion in code reviews, we perform the consistency check within the topics and deduction of more generic topics, as recommended by Zimmermann [31], as well as a consistency check across **RQs** (*i.e.*, reasons, impacts, and coping strategies) and emergent dimensions.

IV. RESULTS

We discuss the application of the research method in practice (Section IV-A), and analyze similarity between the responses received at each one of the survey iterations (Section IV-B). Then, we present the demographics results from the survey (Section IV-C), and discuss reasons for confusion (**RQ1**, Section IV-D), its impact (**RQ2**, Section IV-E), and the strategies employed to cope with it (**RQ3**, Section IV-F).

A. Implementation of Approach

The implementation of the approach designed in Section III is shown in Figure 2. Individuals involved in the card sorting are graduate students in computer science or researchers.

First, following the iterative approach we have performed three iterations since saturation has been reached. Among the 4,645 emails sent during the first iteration, 880 emails have bounced; hence, 17 valid responses correspond to the response rate 0.45%. Such response rate was unexpected[3] and might have been caused by presence of inactive members or one-time-contributors [42]. For the second and the third survey rounds, the number of responses are 24 and 13 respectively; the response rate could not be computed.

The open card sorting of the first survey resulted in 52 topics related to the reasons (25), impacts (14) and coping strategies for confusion (13). The closed card sorting of the second survey resulted in three additional topics: two for impacts and one for the coping strategies. Finally, the closed card sorting of the third survey resulted in no new topics. The open card sorting on the general comments resulted in 16 topics related only to the reasons for confusion, *i.e.*, no topics related to the impacts and coping strategies appeared. Then, the closed card sorting on the inline comments resulted in no new topics.

During the triangulation, we verified that what developers said about the reasons for confusion (survey) has a little agreement with what developers did in the code review comments. Only 6 topics were found both among the survey answers and code review comments, 19 topics appeared only in the survey and 10 topics—in the code review comments. Thus, we decided to conduct another card sorting on the divergent

[2]Both methods are available as functions in the R package *vegan*. ANOSIM has been implemented by Jari Oksanen, with a help from Peter R. Minchin. ADONIS has been implemented by Martin Henry H. Stevens and adapted to *vegan* by Jari Oksanen.

[3]The common response rates in Software Engineering range between 15% and 20% [37], [38], [39], [40] and sometimes much higher response rates are reported [41].

29 topics. This time, since it was an open card sorting, from the cards belonging to divergent topics we identified 42 topics. As the last step, we finalized the comprehensive framework and obtained a total of 57 topics related to reasons (30), impacts (14), and coping strategies (13). After finalizing the topics we observe that 70% (21/30) of them have cards both from the surveys and from the review comments. Moreover, the shared topics cover the lion's share of the cards: 94.9% of the survey cards and 90.7% of the code review comments' cards.

As explained above, using axial coding we identified the following dimensions, common for answers to the three **RQs**: **review process** (18 topics): the code review process, including issues that affect the review duration; **artifact** (15 topics): the system prior to change, code change itself and its documentation or the system after change; **developer** (15 topics): topics regarding the person implementing or reviewing the change; **link** (9 topics): the connection between developers and artifacts, *e.g.*, when a developer indicates that they do not understand the code. Examples of topics of different dimensions can be found in Sections IV-D, IV-E and IV-F.

B. Analysis of Similarity of the Surveys' Results

First, we verified the similarity of the second and third surveys. Since both were published on FACEBOOK and TWITTER, we expect the values to be similar, *i.e.*, respondents to represent the same population. Using both ANOSIM ($R = -0.0171$ and p-value = 0.542) and ADONIS (p-value = 0.975) we could not observe statistically significant differences between the groups, *i.e.*, the answers can be grouped together. Then, we checked the similarity between the answers to the first survey (ANDROID developers) and the answers to the second and the third surveys taken together. The results of the ANOSIM analysis, $R = 0.126$ and p-value = 0.01, showed that the difference between the groups is statistically significant. However, the low R means that the groups are not so different (values closer to 1 mean more of a difference between samples), *i.e.*, the overlap between the surveys is quite high. This observation is confirmed by the outcome of the ADONIS test: the p-value = 0.191 is above the commonly used threshold of statistical significance (0.05). Based on those results, we conclude that the respondents represent the same population of developers and report the results of all three surveys together.

C. Demographics of the Survey Respondents

The respondents are experienced *code reviewers*, 80% (38 of 47 respondents that answered questions about demographics) have more than two years of experience reviewing code changes. The experience of our population as *developers*, *i.e.*, authoring code changes, is even higher: 93% (44 respondents) have been developing for more than two years. The number of years of experience as *developers* is higher than the number of years of experience as *reviewers*: this is expected because reviewing tasks are usually assigned only to more experienced individuals [43]. Respondents are active in submitting changes for review, and even more active in reviewing changes: almost 49% (23 developers) submit code reviews several times a week, while for reviewing this percentages reaches 72% (34).

The frequencies with which code change authors and code reviewers experience confusion are summarized in Figure 3. On the one hand, when reviewing code changes, about 41% (20) of the respondents feel confusion at least half of the time, and only 10% (5) do not feel confusion. On the other hand, when authoring code changes only 12% (6) of the respondents feel confusion at least half of the time, and 35% (17) of the respondents do not feel confusion. Comparing the figures we conclude that confusion when reviewing is very common, and that developers are more often confused when reviewing changes submitted by others as opposed to when authoring the change themselves. We also applied the χ^2 test to check whether experience influences frequency of confusion being experienced. The test was not able to detect differences between more and less experienced developers in terms of frequency of confusion being experienced as a developer, nor between more and less experienced reviewers in terms of frequency of confusion being experienced as a reviewer ($p \simeq 0.26$ and 0.09, respectively).

D. RQ1. What Are the Reasons for Confusion in Code Reviews?

We found 30 reasons for confusion in code review (see Table II). They are spread over all the dimensions, with the artifact and review process being the most prevalent.

There are seven reasons for confusion related to the code review process. The most common is *organization of work* which comprises reasons such as unclear commit message (*e.g.*, "*when the description of the pull request is not clear*", R50), the status of the change (*e.g.*, " *I'm unsure about the status of your parallel move changes. Is this one ready to be reviewed? [...]*")[4], or the change addressing multiple issues (*e.g.*, "*change does more than one things*", R31). The second and third reasons most cited are, respectively, confusion about the tools, *e.g.*, "*I don't know why the rebases are causing new CLs*",[5], and the need of the code change, *e.g.*, "*If I understand correctly, this change might not be relevant any more*".[6]

The artifact dimension it is the largest group with 11 topics related to the reasons for confusion. The most popular is the absence of the change rationale, *e.g.*, "*I do not fully understand why the code is being modified*" (R20). Discussion of the solution related to non-functional aspects of the artifact is the second largest topic and it comprises reasons such as poor code readability (*e.g.*, "*Poorly implemented code*" (R43)), and performance (*e.g.*, "*is this true? i can't tell any difference in transfer speed with or without this patch. i still get roughly these numbers from "adb sync" a -B build of bionic: [...]*").[7] The third most frequent reason indicates that developers experience confusion when *unsure about the system*

[4]https://android-review.googlesource.com/c/132581
[5]https://android-review.googlesource.com/c/71976
[6]https://android-review.googlesource.com/c/33140
[7]https://android-review.googlesource.com/c/91510

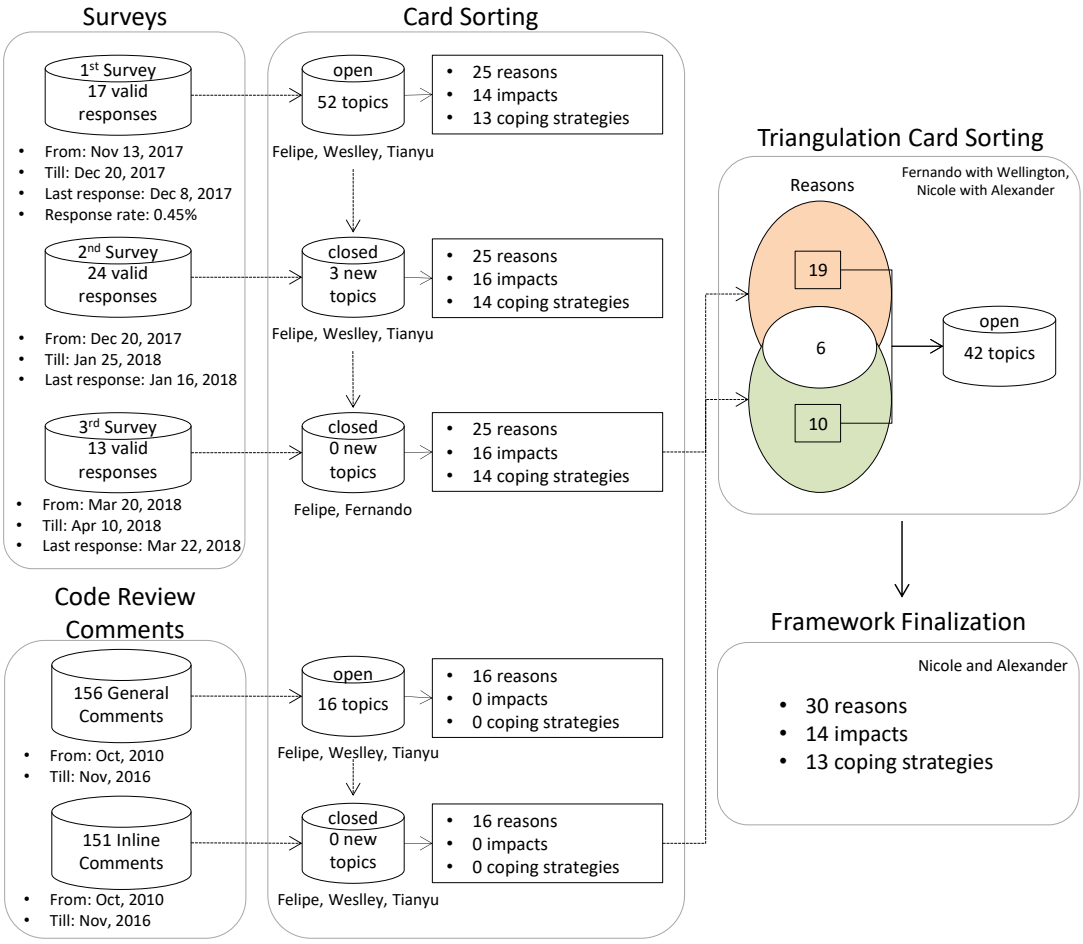

Fig. 2. Implementation of the approach: three survey rounds, general and inline comments, the triangulation, and finalization rounds.

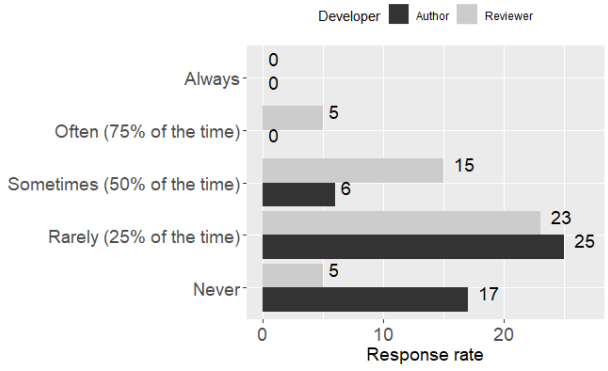

Fig. 3. Frequency of confusion for developers and reviewers.

behavior, *e.g.*, "*what is the difference between this path (false == unresolved) and the unresolved path below. [...]*".[8]

Six reasons for confusion are related to the developer dimension. *Disagreement* among the developers is the prevalent topic, *e.g.*, "*[...] If actual change has a big difference from my expectation, I am confused.*" (R11). The second most cited reason is the misunderstanding of the message's intention, *e.g.*, "*Sometimes I don't understand general meaning (need to read several times to understand what person means)*" (R13).

Six reasons are related to the link between the developer and the artifact. The most popular one is the *lack of familiarity with existing code*, *e.g.*, "*Lack of knowledge about the code that's being modified.*" (R37) followed by the *lack of programming skills*, *e.g.*, "*sometimes I'm confused because missing some programming*" (R13), and the *lack of understanding of the problem*, *e.g.*, "*I'm embarrassed to admit it, but I still don't understand this bug.*".[9]

RQ1 Summary: We found a total of 30 reasons for confusion. The most prevalent are *missing rationale*, *discussion of the solution: non-functional*, and *lack of familiarity with existing code*. We observe that tools (code review, issue tracker, and version control) and communication issues, such as disagreement or ambiguity in communicative intentions, may also cause confusion during code reviews.

[8]https://android-review.googlesource.com/c/83350

[9]https://android-review.googlesource.com/c/170280

TABLE II
THE REASONS, IMPACTS AND COPING STRATEGIES DEVELOPERS USE TO DEAL WITH CONFUSION; IN THE PARENTHESIS ARE THE NUMBERS OF CARDS.

	Review process *18 topics (120)*	**Artifact** *15 topics (300)*	**Developer** *15 topics (124)*	**Link** *9 topics (177)*
Reasons *30 topics (507)*	Organization of work (17) Issue tracker, version control (7) Unnecessary change (6) Not enough time (3) Dependency between changes (3) Code ownership (2) Community norms (2)	Missing rationale (66) Discussion of the solution: non-functional (49) Unsure about system behavior (37) Lack of documentation (29) Discussion of the solution: strategy (29) Long, complex change (25) Lack of context (19) Discussion of the solution: correctness (14) Impact of change (11) Irreproducible bug (6) Lack of tests (5)	Disagreement (18) Communicative intention (9) Language issues (3) Propagation of confusion (3) Fatigue (1) Noisy work environment (1)	Lack of familiarity with existing code (47) Lack of programming skills (40) Lack of understanding of the problem (21) Lack of understanding of the change (17) Lack of familiarity with the technology (14) Lack of knowledge about the process (3)
Impacts *14 topics (98)*	Delaying (31) Decreased review quality (11) Additional discussions (11) Blind approval (8) Review rejection (4) Increased development effort (4) Assignment to other reviewers (2)	Better solution (1) Incorrect solution (1)	Decreased confidence (10) Abandonment (6) Frustration (5) Anger (2) Propagation of confusion (2)	
Coping strategies *13 topics (116)*	Improved organization of work (5) Delaying (2) Assignment to other reviewers (1) Blind approval (1)	Small, clear changes (4) Improved documentation (4)	Information requests (36) Off-line discussions (12) Providing/accepting suggestions (10) Disagreement resolution (6)	Improved familiarity with existing code (28) Testing the change (5) Improved familiarity with the technology (2)

E. RQ2. What Are the Impacts of Confusion in Code Reviews?

The total number of topics related to the impacts of confusion is 14 (see Table II). They are related to the dimensions of the review process, artifact, and developer. There was no topic related to the link between the developer and the artifact.

We found seven impacts of confusion related to the code review process. *Delaying* the merge decision is the most popular impact, *e.g.*, "*The review takes longer than it should*" (R46). The second and third most cited impact are that confusion makes the code review quality decrease, *e.g.*, "*Well I can't give a high quality code review if I don't understand what I am looking at*" (R5), and an increase in the number of messages exchanged during the discussion, *e.g.*, "*Code reviews take longer as there's additional back and forth*" (R1). One interesting impact of confusion is the blind approval of the code change by the developer, even without understanding it, *e.g.*, "*Blindly approve the change and hope your coworker knows what they're doing (it is clearly the worst; that's how serious bugs end up in production)*" (R16). Confusion may also lead to developers to just reject a code change, *e.g.*, "*I'm definitely much more likely to reject a 'confusing' code review. Good code, in my experience, is usually not confusing*" (R36).

There are only two impacts of confusion related to the artifact itself. First, the developer may find a better solution because of the confusion, *e.g.*, "*It has not only bad impact but also good impact. Sometimes I can encounter a better solution than my thought*" (R11). Second, the code change might be approved with bugs, as the reviewer is not be able to review it properly due confusion, *e.g.*, "*Sometimes repeated code is committed or even a wrong functionality*" (R24). The *incorrect solution* impact is related to *decrease review quality*, however, the perspective is of the code change containing a bug in production rather than of the reviewing process.

Finally, there are four impacts of confusion related to the developer. The most quoted impact is the decrease of self confidence, either by the author, *e.g.*, "*I can't be confident my change is correct*" (R38), or by the reviewer, *e.g.*, "*I feel less confident about approving it*" (R48). Another impact is the developer giving up, abandoning a code change instead of accounting for the reviewer's comments, *e.g.*, "*other times I just give up*" (R14), or leave the project, *e.g.*, "*dissociated myself a little from the codebase internally*" (R14). We also found emotions being triggered by confusion, such as anger (*e.g.*, "*It pissed me off*", R3) and frustration (*e.g.*, "*Cannot be an effective reviewer—can replace me with a lemur*", R40). And finally, confusion can be contagious, *e.g.*, "*It often causes confusion spreading to other reviewers*" (R12).

> ***RQ2 Summary:*** We identified 14 different impacts of confusion in code reviews. The most common are *delaying*, *decrease of review quality*, and *additional discussions*. Some developers blindly approve the code change, regardless the correctness of it; other impacts include *frustration*, *abandonment* and *decreased confidence*.

F. RQ3. How Do Developers Cope with Confusion?

We found 13 topics describing the strategies developers use to deal with confusion in code reviews. Four of them are related to the review process. The most common is to *improve the organization of work*, such as making clearer commit messages, *e.g.*, "*Leave comments on the files with the main changes*" (R50). It is followed by spending more time and *delaying* the code review, *e.g.*, "*I need to spend much more time*" (R13). Assigning other reviewers is also a strategy adopted by developer, *e.g.*, "*Sometimes I completely defer to other reviewers*" (R48). Interestingly, *blind approval* is also a strategy developers use to cope with confusion, *i.e.*, it is not just an impact, *e.g.*, "*assume the best, (of the change)*" (R34).

Two strategies are related to the artifact. Developers make the code change smaller, *e.g.*, "*Also I ask large changes to be*

broken into smaller" (R31), and clearer, *e.g.*, "*Try to make the actual code change clear*" (R12). They also improve the documentation by adding code comments, *e.g.*, "*A good description in the commit message describing the bug and the method used to fix the bug is also helpful for reviewers*" (R5).

The dimension with the most quotes is related to the developers themselves. Requesting for information on the code review tool itself is the most quoted among developers, *e.g.*, "*Put comment and ask submitter to explain unclear points*" (R15). Developers also take the discussions off-line, *i.e.*, using other means to reach their peers, *e.g.*, "*schedule meetings*" (R50) or "*ask in person*" (R1). Providing and accepting suggestions is also mentioned as a good way to cope with confusion. It includes strategies such as being open minded to the comments of their peers, *e.g.*, "*Being open to critical review comments*" (R12), and providing polite criticism, *e.g.*, "*Trying to be 'a nice person'. Gently criticizing the code*" (R3). The use of criticism by developers in code reviews was also found by Ebert *et al.* [19], but their study focused on the intention of questions in code reviews. Disagreement resolution is also a good strategy to cope with confusion, *e.g.*, "*I try to explain the reasoning behind the decisions/assumptions I made*" (R31).

Regarding the link between the developer and the artifact, there are three strategies developers use to cope with confusion. Firstly, to study the code or the documentation, *e.g.*, "*It forces me to dig deeper and learn more about the code module to make sure that my understanding is correct (or wrong)*" (R12), and "*Read requirements documentation*" (R24). Secondly, to test the code change, *e.g.*, "*play with the code*" (R9). Finally, developers also use external sources to improve their knowledge about the technology, *e.g.*, "*Sometimes further research on the web [...]*" (R25).

RQ3 Summary: We have identified 13 coping strategies. Common strategies include *information requests*, *improved familiarity with the existing code*, and *off-line discussions*.

V. Discussion and Implications

The main contribution of our study is the empirically-driven comprehensive framework of reasons, impacts, and coping strategies for confusion. Our study suggests practical, actionable implications for the tool builders (Section V-B) as well as insights and suggestions for researchers (Section V-C).

A. Discussion

Confusion is an inherent part of human problem-solving, which normally arises from information and goal-oriented assessment of situations [44], [45]. Evidence from psychology research demonstrates that individuals engaged in a complex cognitive task continually assimilate new information into existing knowledge structures in order to achieve completion of their tasks [46]. Any possible mismatch between expectations and (lack of) previous knowledge might be responsible for confusion. Indeed, 92 cards (over 290) for reasons for confusion are associated to 'discussion' about the *artifact*, *i.e.*, *discussion of the solution: non-functional, strategy*, and *correctness*. Symmetrically, common coping strategies involve *information requests* and *off-line discussions*.

Confusion might trigger in individuals the experience of negative emotions, depending on the context, the amount of mismatch between expectation and reality, and the extent to which completion of tasks is threatened [47]. This is in line with the observation of *decreased confidence*, *abandonment*, and such emotions as *frustration* and *anger*, among the impacts of confusion. Early detection of confusion [17] becomes crucial for preventing contributors' burnout and loss of productivity [48]. Such situations might even cause the abandonment of the project, which is undesirable as it eventually leads to undesired turnover; since developers focusing on documentation are more susceptible to turnover than those working on code [49], project abandonment is likely to exacerbate *lack of documentation* further increasing confusion within the project. A possible antidote to negative emotions, identified as a coping strategy by the survey respondents, is being open minded when *providing and accepting suggestions*. This attitude is often mandated by codes of conducts adopted by open source projects [50], requiring, *e.g.*, to "gracefully accept constructive criticism" and "be respectful of differing viewpoints and experiences". One should be aware, however, that imposing such rules might lead to emotional labor [51].

B. Implications for Tool Builders

Code reviews are supported by such tools as Gerrit. Currently the only feature of Gerrit that we can relate to confusion reduction is flagging large code changes. Indeed, *large, complex changes* are among the most popular reasons for confusion in our framework. As a strategy to deal with this issue, developers suggest to split big changes into smaller, clearer ones. This is consistent with the earlier findings [11], envisioning the emergence of tools enabling early detection of splittable changes, *i.e.*, before the pull request is submitted, in order to both avoid spending additional time in identifying such patches and asking the author re-work the change to reduce its complexity and likelihood of the discussion [52].

Based on our results, further tool improvements can be envisioned. We observe that the second most popular coping strategy is to *improve familiarity with existing code*. The burden of this task might be reduced if code review tools could provide the task context [16]. Similarly, a summary of the change [53], [54] could be beneficial to overcome confusion due to *lack of understanding of the change*. *Organization of work* can be improved by tools capable of automatically generating commit messages [55], [56], disentangling commits performing multiple tasks [57] and combining multiple commits performing one task [58]. Furthermore, information retrieval tools able to understand written design discussions occurring in pull requests [59] could be integrated into code review tools, to extract rationale, implicit knowledge, and other contextual information otherwise not available during the review process. Integrating such information into the documentation might prevent confusion due to a *missing rationale*, *lack of context*, or doubt about *necessity of a change*.

Another reason for confusion is the difficulty in assessing the *impact of the change*. Integration of change impact analysis tools might be thus beneficial. Similarly, integration of tools assessing the test coverage of a change might keep developers from committing changes with low test coverage, thus avoiding confusion due to *lack of tests*. Information about test coverage could be also integrated in the pull request, *i.e.*, by mean of badges as done by tools as COVERALLS.[10]

Developers also report *off-line discussions* as a strategy to quickly *resolve disagreement* as well as ambiguities in *communicative intentions*. This evidence is consistent with findings from previous research by Pascarella *et al.* [11], who also envision the integration of synchronous communication functions into code review tools to enable traceability of decisions and explanation provided and allow their integration into documentation for future reference.

Going beyond code review tools, developers experience confusion in using issue tracking or version control systems. Hence these tools can be improved to facilitate their usage.

C. Implications for Researchers

In our framework, observed reasons for confusion are far more than coping strategies. Indeed, strategies are derived by the analysis of developers' self-report in the survey and represent what they already do to deal with confusion and uncertainty. Conversely, reasons for confusion are defined based on the analysis of both survey responses and developers' comments during code review. Of course, a symmetry is observed for those reason-strategy couples addressing the same cause of confusion, *e.g.*, *small, clear changes* are offered as a solution for confusion due to *long, complex changes*, *information requests* can address difficulties in understanding the others' *communicative intentions*, *lack of familiarity with existing code* is addressed by studying the code and its documentation (*improved familiarity with existing code*). A significant amount of reasons, though, are not addressed. Addressing these topics with follow-up studies might be beneficial identifying what *could* be done and how, in order to improve code review, in addition to what is already done and reported by developers.

We observed that the assignment of other reviewers in the discussion is either an impact and a strategy developers adopt to deal with confusion. Thus, confusion can be beneficial as contribute to decrease the number of bugs in the software as more reviewers tend avoid bugs [2], [60]. Moreover, the inclusion of more people in the code review increases their awareness of the code change, *i.e.*, confusion resolution contributes to knowledge sharing. However, involving additional reviewers induces additional workload, while reviewers expertise might already be scarce. Indeed, Ruangwan *et al.*have observed that 16%-66% review requests have at least one invited reviewer who did not respond to the invitation [61]. Moreover, involvement of additional reviewers might lead to further disagreement between them, leading to inability to complete the software engineering task [62]. This is why research should consider both early detection of confusion alleviating the need of involving additional reviewers, and better recommendation of reviewers for a given code change [63].

We have observed that presence of confusion causes developers to move discussion off-line, *i.e.*, outside the code review tool. This finding is similar to earlier observations [64] suggesting that despite the omnipresence of platforms such as GERRIT and GITHUB fostering transparency of software development, cases experienced as confusing sometimes remain invisible. This finding might threaten validity of previously published studies of code reviews that have been solely based on mining digital trace data. More comprehensive research methods should therefore be sought.

We have observed that the most popular coping strategy is *information requests*. So far, little research was conducted on information needs in code reviews [19], [11]. Ebert *et al.* [19] show that almost half of the developers' questions have the intention to seek for some kind of information, such as confirmation, information, rationale, clarification, and opinion. The information needs expressed by developers during code reviews [11] are closely related to the reasons for confusion in Table II. Indeed, *suitability of an alternative solution* [11] is related to the *discussion of the solution strategy*, *splittable* [11] to *long, complex change* and *specialized expertise* to *lack of familiarity with technology*. We believe, therefore, that confusion may trigger both of the requests for information [19] and the information needs [11]. Hence, we see a clear venue for researchers to understand better both the information needed during a code review and the ways developers aim at obtaining it. Using this understanding one should try to automatize supplying the relevant information to resolve confusion.

VI. THREATS TO VALIDITY

As any empirical study, our work is subject to several threats of validity. **Construct validity** is related to the relation between the concept being studied and its operationalisation. In particular, it is related to the risk of respondents misinterpreting the survey questions. To reduce this risk we included our own definition of confusion and requested the respondents to confirm that they understood it. For the same reason, we always anchored the frequency questions and adhered to well-known survey design recommendations [26], [27], [28], [29]. **Internal validity** pertains to inferring conclusions from the data collected. The card sorting adopted in our work is inherently subjective because of the necessity to interpret text. To reduce subjectivity every card sorting step has been carried out by several researchers. Moreover, to assure the completeness of the topics related to the reasons, impacts and confusion coping strategies we conducted several survey iterations till the data saturation has been achieved, and augmented the insights from the surveys with those from the code review comments. **External validity** is related to the generalizability of the conclusions beyond the specific context of the study. Our first survey targeted only a single project: ANDROID. However, the second and the third ones targeted a general software developer population. Statistical analysis has not revealed any differences

[10]https://coveralls.io/

between the respondents of the different surveys suggesting that the answers obtained are likely to reflect opinions of the code review participants, in general. To complement the surveys we consider 307 code review comments from GERRIT. While the functionality of GERRIT is typical for most modern code review tools, developers using more advanced code review tools do not necessarily experience confusion in the same way. For instance, COLLABORATOR[11] supports custom templates and checklists, that if properly configured might require the change authors to indicate rationale of their change, reducing the importance of "missing rationale" from Table II.

VII. RELATED WORK

A. Code Review

Code review has been the focus of a plethora of studies [2], [9], [14], [65], [66], [67], [68], [69], [70], [43]. Bacchelli and Bird [9] analyzed the expectations, outcomes, and challenges of developers while conducting code reviews at MICROSOFT. They found that the developers' top motivations in code reviews are to find defects, but the most common outcome is actually code improvement, and the main challenge of code review is understanding the patch set. Tao *et al.* [14] investigated how the understanding of patch sets affects the development process and show that the most important information for understanding the code review is the rationale of the code change. Bavota and Russo [2] show that code changes that were not reviewed have over two times more chances of introducing defects than reviewed ones, and the reviewed code changes have a substantially higher readability when compared to the ones not reviewed. Kononenko *et al.* [65] investigated the quality of code reviews and found that 54% of the reviewed changes introduced defects in the system. Hentschel *et al.* [66] provided evidence for increased effectiveness when a symbolic execution tree view is used during code reviews. Pascarella *et al.* [11] investigated, by analyzing code review comments, what information reviewers need to perform a proper code review. They found seven high-level information needs, such as the correct understanding of the code change, and rationale.

B. Confusion

Confusion has been studied before, also in relation with complex cognitive tasks [18], [46]. Approaches to automatic identification of confusion have been recently developed, based on natural language processing [72], [73], [17]. Yang *et al.* [72] used textual content of comments from a forum and its clickstream data to automatically identify posts that express confusion. Their model to identify confusion comprises questions, users' click patterns, and users' linguistic features based on LIWC[12] words. Jean *et al.* [73] proposed an approach to detect uncertain expressions based on the statistical analysis of syntactic and lexical features. Ebert *et al.* [17] assessed the feasibility of automatic recognition of confusion in code review comments based on linguistic features. They assessed the performance of several classifiers based on supervised training, using a gold standard of 800 comments manually labeled as indicating or not a developer's confusion.

Confusion-related phenomena have been recently investigated in code reviews. Ram *et al.* [74] aimed to obtain an empirical understanding of what makes a code change easier to review. They found that reviewability is affected by several factors, such as the change description, size, and coherent commit history. Barik *et al.* [75] conducted an eye tracking study to understand how developers use compiler error messages. They found that the difficulty experienced by developers while reading error messages is a significant predictor of task correctness. Gopstein *et al.* [76] introduced the term *atom of confusion* which is the smallest code pattern that can reliably cause confusion in a developer. Through a controlled experiment with developers, they studied the prevalence and significance of the atoms of confusion in real projects. They report a strong correlation between these confusing patterns and bug-fix commits, as well as a tendency for confusing patterns to be eventually commented.

To the best of our knowledge, this is the first study that aims at building a comprehensive framework of what make developers confused during code reviews, their impacts and what strategies do developers implement to overcome confusion.

VIII. CONCLUSIONS

We built a comprehensive framework for confusion in code reviews including its reasons, impacts, and the coping strategies adopted by developers. We used a concurrent triangulation strategy combining a developer's survey and the analysis of code review comments. We found 30 reasons for confusion, with the most common ones being *missing rationale*, *discussion of the solution: non-functional*, and *lack of familiarity with existing code*. Among the 14 impacts of confusion, the prevalent are *delaying*, *decrease of review quality*, and *additional discussions*. Finally, developers employ 13 strategies to cope with confusion, such as *information requests*, and *off-line discussions*.

Our study has several implications for both tool builders and researchers. Code review tools could be improved by integrating information that can reduce confusion, *e.g.*, related to rationale, test coverage or impact of the change. Researchers should investigate possible relations between confusion and information needs, as well as between confusion and migration of code review discussions to off-line channels.

ACKNOWLEDGMENTS

This research was partially funded by CNPq/Brazil (304755/2014-1, 406308/2016-0, 465614/2014-0), CAPES/Brazil (PDSE-88881.132399/2016-01), FACEPE/Brazil (APQ-0839-1.03/14, 0388-1.03/14, 0791-1.03/13), and by the project "EmoQuest", funded by the Italian Ministry for Education, University and Research under the SIR program. We are very grateful to Tianyu Liu, Weslley Torres and Wellington Oliveira for their help with the card sorting.

[11] https://smartbear.com/product/collaborator/overview/

[12] https://liwc.wpengine.com

REFERENCES

[1] Y. Tao and S. Kim, "Partitioning composite code changes to facilitate code review," in *MSR*. IEEE, 2015, pp. 180–190.

[2] G. Bavota and B. Russo, "Four eyes are better than two: On the impact of code reviews on software quality," in *ICSME*, 2015, pp. 81–90.

[3] B. Boehm and V. R. Basili, "Top 10 list [software development]," *Computer*, vol. 34, no. 1, pp. 135–137, 2001.

[4] M. V. Mäntylä and C. Lassenius, "What types of defects are really discovered in code reviews?" *TSE*, vol. 35, no. 3, pp. 430–448, 2009.

[5] M. Barnett, C. Bird, J. Brunet, and S. K. Lahiri, "Helping developers help themselves: Automatic decomposition of code review changesets," in *ICSE*. IEEE, 2015, pp. 134–144.

[6] J. Cohen, S. Teleki, and E. Brown, *Best Kept Secrets of Peer Code Review*. Smart Bear Inc., 2006.

[7] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "An empirical study of the impact of modern code review practices on software quality," *ESE*, pp. 1–44, 2015.

[8] T. Pangsakulyanont, P. Thongtanunam, D. Port, and H. Iida, "Assessing MCR discussion usefulness using semantic similarity," in *Empirical Software Engineering in Practice (IWESEP), 2014 6th International Workshop on*, Nov 2014, pp. 49–54.

[9] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *ICSE*. IEEE, 2013, pp. 712–721.

[10] R. Morales, S. McIntosh, and F. Khomh, "Do code review practices impact design quality? a case study of the qt, vtk, and itk projects," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, March 2015, pp. 171–180.

[11] L. Pascarella, D. Spadini, F. Palomba, M. Bruntik, and A. Bacchelli, "Information needs in contemporary code review," to appear in Proceedings of the ACM Conference on Computer Supported Cooperative Work, ser. CSCW '18, 2018.

[12] M. Greiler, "On to code review: Lessons learned @ microsoft," 2016, keynote for QUATIC 2016 - the 10th International Conference on the Quality of Information and Communication Technology. [Online]. Available: https://pt.slideshare.net/mgreiler/on-to-code-review-lessons-learned-at-microsoft

[13] A. Bosu, J. C. Carver, C. Bird, J. Orbeck, and C. Chockley, "Process aspects and social dynamics of contemporary code review: Insights from open source development and industrial practice at microsoft," *IEEE Transactions on Software Engineering*, vol. 43, no. 1, pp. 56–75, Jan 2017.

[14] Y. Tao, Y. Dang, T. Xie, D. Zhang, and S. Kim, "How do software engineers understand code changes?: An exploratory study in industry," in *FSE*. ACM, 2012, pp. 51:1–51:11.

[15] A. Sutherland and G. Venolia, "Can peer code reviews be exploited for later information needs?" in *ICSE-Companion*, 2009, pp. 259–262.

[16] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining mental models: A study of developer work habits," in *ICSE*. New York, NY, USA: ACM, 2006, pp. 492–501.

[17] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik, "Confusion detection in code reviews," in *ICSME*, 2017, pp. 549–553.

[18] S. D'Mello and A. Graesser, "Confusion and its dynamics during device comprehension with breakdown scenarios," *Acta Psychologica*, vol. 151, pp. 106–116, 2014.

[19] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik, "Communicative intention in code review questions," in *ICSME*, 2018.

[20] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, *Selecting Empirical Methods for Software Engineering Research*. London: Springer London, 2008, pp. 285–311. [Online]. Available: https://doi.org/10.1007/978-1-84800-044-5_11

[21] M. E. Fagan, "Design and code inspections to reduce errors in program development," *IBM Syst. J.*, vol. 15, no. 3, pp. 182–211, Sep. 1976. [Online]. Available: http://dx.doi.org/10.1147/sj.153.0182

[22] E. S. Raymond, *The Cathedral and the Bazaar*, 1st ed., T. O'Reilly, Ed. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 1999.

[23] W. F. Tichy, "Rcs - a system for version control," *Software: Practice and Experience*, vol. 15, pp. 637–654, 1985.

[24] B. G. Glaser and A. L. Strauss, *The Discovery of Grounded Theory: Strategies for Qualitative Research*. New York, NY: Aldine de Gruyter, 1967.

[25] K.-J. Stol, P. Ralph, and B. Fitzgerald, "Grounded theory in software engineering research: A critical review and guidelines," in *ICSE*, May 2016, pp. 120–131.

[26] R. M. Groves, F. J. Fowler, M. P. Couper, J. M. Lepkowski, E. Singer, and R. Tourangeau, *Survey Methodology*, 2nd ed. Wiley, 2009.

[27] B. Kitchenham and S. L. Pfleeger, "Personal opinion surveys," in *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. I. K. Sjoberg, Eds., 2008, pp. 63–92.

[28] J. Singer and N. G. Vinson, "Ethical issues in empirical studies of software engineering," *IEEE Transactions on Software Engineering*, vol. 28, no. 12, pp. 1171–1180, Dec 2002.

[29] C. M. Steele and J. Aronson, "Stereotype threat and the intellectual test performance of african americans." *Journal of personality and social psychology*, vol. 69 5, pp. 797–811, 1995.

[30] W. H. Foddy, *Constructing questions for interviews and questionnaires: theory and practice in social research*. Cambridge University Press Cambridge, UK ; New York, NY, USA, 1993.

[31] T. Zimmermann, "Card-sorting: From text to themes," in *Perspectives on Data Science for Software Engineering*, T. Menzies, L. Williams, and T. Zimmermann, Eds. Boston: Morgan Kaufmann, 2016, pp. 137 – 141. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128042069000271

[32] D. Finfgeld-Connett, "Use of content analysis to conduct knowledge-building and theory-generating qualitative systematic reviews," *Qualitative Research*, vol. 14, no. 3, pp. 341–352, 2014. [Online]. Available: https://doi.org/10.1177/1468794113481790

[33] P. Lenberg, R. Feldt, L. G. W. Tengberg, I. Tidefors, and D. Graziotin, "Behavioral software engineering - guidelines for qualitative studies," *CoRR*, vol. abs/1712.08341, 2017. [Online]. Available: http://arxiv.org/abs/1712.08341

[34] K. R. Clarke, "Non-parametric multivariate analysis of changes in community structure," *Australian Journal of Ecology*, vol. 18, pp. 117–143, 1993.

[35] M. J. Anderson, "A new method for non-parametric multivariate analysis of variance," *Austral Ecology*, vol. 26, no. 1, pp. 32–46, 2001. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1442-9993.2001.01070.pp.x

[36] B. H. McArdle and M. J. Anderson, "Fitting multivariate models to community data: A comment on distance-based redundancy analysis," *Ecology*, vol. 82, no. 1, pp. 290–297, 2001.

[37] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, D. Poshyvanyk, and A. De Lucia, "Mining version histories for detecting code smells," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 462–489, May 2015.

[38] B. Vasilescu, V. Filkov, and A. Serebrenik, "Perceptions of diversity on git hub: A user survey," in *2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering*, May 2015, pp. 50–56.

[39] B. Vasilescu, D. Posnett, B. Ray, M. G. J. van den Brand, A. Serebrenik, P. Devanbu, and V. Filkov, "Gender and tenure diversity in github teams," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, ser. CHI '15. New York, NY, USA: ACM, 2015, pp. 3789–3798. [Online]. Available: http://doi.acm.org/10.1145/2702123.2702549

[40] H. S. Qiu, A. Nolte, A. Brown, A. Serebrenik, and B. Vasilescu, "Going farther together: The impact of social capital on sustained participation in open source," in *ICSE*. IEEE, 2019.

[41] F. Palomba, D. A. Tamburri, A. Serebrenik, A. Zaidman, F. A. Fontana, and R. Oliveto, "How do community smells influence code smells?" in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, ser. ICSE '18. New York, NY, USA: ACM, 2018, pp. 240–241. [Online]. Available: http://doi.acm.org/10.1145/3183440.3194950

[42] A. Lee, J. C. Carver, and A. Bosu, "Understanding the impressions, motivations, and barriers of one time code contributors to FLOSS projects: a survey," in *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017*, S. Uchitel, A. Orso, and M. P. Robillard, Eds. IEEE / ACM, 2017, pp. 187–197.

[43] P. van Wesel, B. Lin, G. Robles, and A. Serebrenik, "Reviewing career paths of the openstack developers," in *ICSME*. IEEE Computer Society, 2017, pp. 544–548.

[44] G. Mandler, *Mind and Body: Psychology of Emotion and Stress*. W.W. Norton and Company, New York, 1984.

[45] ——, *Interruption (discrepancy) theory: review and extensions*. Wiley, Chichester, 1990, pp. 13–32.

[46] S. D'Mello, B. Lehman, R. Pekrun, and A. Graesser, "Confusion can be beneficial for learning," *Learning and Instruction*, vol. 29, pp. 153 – 170, 2014.

[47] N. Stein and L. Levine, *Making sense out of emotion*. Erlbaum, Hillsdale, NJ, 1991, pp. 295–322.

[48] M. Mäntylä, B. Adams, G. Destefanis, D. Graziotin, and M. Ortu, "Mining valence, arousal, and dominance: Possibilities for detecting burnout and productivity?" in *Proceedings of the 13th International Conference on Mining Software Repositories*, ser. MSR '16. New York, NY, USA: ACM, 2016, pp. 247–258. [Online]. Available: http://doi.acm.org/10.1145/2901739.2901752

[49] B. Lin, G. Robles, and A. Serebrenik, "Developer turnover in global, industrial open source projects: Insights from applying survival analysis," in *Proceedings of the 12th International Conference on Global Software Engineering*, ser. ICGSE '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 66–75. [Online]. Available: https://doi.org/10.1109/ICGSE.2017.11

[50] P. Tourani, B. Adams, and A. Serebrenik, "Code of conduct in open source projects," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Feb 2017, pp. 24–33.

[51] A. Serebrenik, "Emotional labor of software engineers," in *Proceedings of the 16th edition of the BElgian-NEtherlands software eVOLution symposium, Antwerp, Belgium, December 4-5, 2017.*, ser. CEUR Workshop Proceedings, S. Demeyer, A. Parsai, G. Laghari, and B. van Bladel, Eds., vol. 2047. CEUR-WS.org, 2017, pp. 1–6.

[52] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida, "Review participation in modern code review - an empirical study of the android, qt, and openstack projects," *Empirical Software Engineering*, vol. 22, no. 2, pp. 768–817, 2017. [Online]. Available: https://doi.org/10.1007/s10664-016-9452-6

[53] S. Panichella, "Summarization techniques for code, change, testing, and user feedback (invited paper)," in *IEEE Workshop on Validation, Analysis and Evolution of Software Tests*, March 2018, pp. 1–5.

[54] P. C. Rigby and M. D. Storey, "Understanding broadcast based peer review on open source software projects," in *ICSE*, R. N. Taylor, H. C. Gall, and N. Medvidovic, Eds. ACM, 2011, pp. 541–550.

[55] M. Linares-Vásquez, L. F. Cortés-Coy, J. Aponte, and D. Poshyvanyk, "Changescribe: A tool for automatically generating commit messages," in *Proceedings of the 37th International Conference on Software Engineering*, vol. 2. Piscataway, NJ, USA: IEEE Press, 2015, pp. 709–712.

[56] Y. Huang, Q. Zheng, X. Chen, Y. Xiong, Z. Liu, and X. Luo, "Mining version control system for automatically generating commit comment," in *Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. Piscataway, NJ, USA: IEEE Press, 2017, pp. 414–423. [Online]. Available: https://doi.org/10.1109/ESEM.2017.56

[57] M. Dias, A. Bacchelli, G. Gousios, D. Cassou, and S. Ducasse, "Untangling fine-grained code changes," in *22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering, SANER 2015, Montreal, QC, Canada, March 2-6, 2015*, Y. Guéhéneuc, B. Adams, and A. Serebrenik, Eds. IEEE Computer Society, 2015, pp. 341–350.

[58] R. Arima, Y. Higo, and S. Kusumoto, "A study on inappropriately partitioned commits: How much and what kinds of ip commits in java projects?" in *Proceedings of the 15th International Conference on Mining Software Repositories*. New York, NY, USA: ACM, 2018, pp. 336–340. [Online]. Available: http://doi.acm.org/10.1145/3196398.3196406

[59] G. Viviani, C. Janik-Jones, M. Famelis, and G. C. Murphy, "The structure of software design discussions," in *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, 2018, pp. 104–107.

[60] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "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*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 192–201. [Online]. Available: http://doi.acm.org/10.1145/2597073.2597076

[61] S. Ruangwan, P. Thongtanunam, A. Ihara, and K. Matsumoto, "The impact of human factors on the participation decision of reviewers in modern code review," *Empirical Software Engineering*, Sep 2018. [Online]. Available: https://doi.org/10.1007/s10664-018-9646-1

[62] F. Palomba, D. A. Tamburri, F. A. Fontana, R. Oliveto, A. Zaidman, and A. Serebrenik, "Beyond technical aspects: How do community smells influence the intensity of code smells?" *IEEE Transactions on Software Engineering*, pp. 1–1, 2019.

[63] P. Thongtanunam, C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida, and K. ichi Matsumoto, "Who should review my code? a file location-based code-reviewer recommendation approach for modern code review," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, March 2015, pp. 141–150.

[64] J. Aranda and G. Venolia, "The secret life of bugs: Going past the errors and omissions in software repositories," in *ICSE*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 298–308.

[65] O. Kononenko, O. Baysal, L. Guerrouj, Y. Cao, and M. W. Godfrey, "Investigating code review quality: Do people and participation matter?" in *ICSME*, 2015, pp. 111–120.

[66] M. Hentschel, R. Hähnle, and R. Bubel, "Can formal methods improve the efficiency of code reviews?" in *IFM*. Springer, 2016, pp. 3–19.

[67] M. Mukadam, C. Bird, and P. C. Rigby, "Gerrit software code review data from android," in *MSR*. IEEE, 2013, pp. 45–48.

[68] K. Hamasaki, R. G. Kula, N. Yoshida, A. E. C. Cruz, K. Fujiwara, and H. Iida, "Who does what during a code review? datasets of oss peer review repositories," in *MSR*. IEEE, 2013, pp. 49–52.

[69] P. Thongtanunam, X. Yang, N. Yoshida, R. G. Kula, A. E. C. Cruz, K. Fujiwara, and H. Iida, "Reda: A web-based visualization tool for analyzing modern code review dataset," in *ICSME*, 2014, pp. 605–608.

[70] X. Yang, R. G. Kula, N. Yoshida, and H. Iida, "Mining the modern code review repositories: A dataset of people, process and product," in *MSR*. ACM, 2016, pp. 460–463.

[71] D. Yang, M. Wen, I. Howley, R. Kraut, and C. Rose, "Exploring the effect of confusion in discussion forums of massive open online courses," in *ACM Conference on Learning @ Scale*. ACM, 2015, pp. 121–130.

[72] P.-A. Jean, S. Harispe, S. Ranwez, P. Bellot, and J. Montmain, "Uncertainty detection in natural language: A probabilistic model," in *International Conference on Web Intelligence, Mining and Semantics*. New York, NY, USA: ACM, 2016, pp. 10:1–10:10.

[73] A. Ram, A. Ashok Sawant, C. Marco, and A. Bacchelli, "What makes a code change easier to review? an empirical investigation on code change reviewability," to appear in 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ser. ESEC/FSE '18, 2018.

[74] T. Barik, J. Smith, K. Lubick, E. Holmes, J. Feng, E. Murphy-Hill, and C. Parnin, "Do developers read compiler error messages?" in *Proceedings of the 39th International Conference on Software Engineering*, ser. ICSE '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 575–585. [Online]. Available: https://doi.org/10.1109/ICSE.2017.59

[75] D. Gopstein, J. Iannacone, Y. Yan, L. DeLong, Y. Zhuang, M. K.-C. Yeh, and J. Cappos, "Understanding misunderstandings in source code," in *ESEC/FSE*. New York, NY, USA: ACM, 2017, pp. 129–139.