Empirical Software Engineering (2025) 30:17
https://doi.org/10.1007/s10664-024-10560-7

A qualitative study on refactorings induced by code review

Flávia Coelho[1] · Nikolaos Tsantalis[2] · Tiago Massoni[3] · Everton L. G. Alves[3]

Accepted: 1 October 2024

Abstract

Modern Code Review (MCR) has become an essential practice in pull-based development since reviewers may provide insights for improvements, such as code refactoring, in Pull Requests (PRs). A recent study explored PRs in light of refactoring-inducement, discovering statistical differences in the number of review comments and discussions between refactoring-inducing and non-refactoring-inducing PRs. In refactoring-inducing PRs, the refactoring edits emerge after the initial commit(s) as a result of review comments and/or spontaneous initiative by the PR developer. In this paper, we report a qualitative study that examines code reviewing-related aspects (review comments and discussion) to characterize code review in refactoring-inducing PRs. We conjecture that code reviews differ in refactoring-inducing from non-refactoring-inducing PRs, which has interesting implications. To investigate our hypothesis, we manually scrutinized 923 review comments and 361 refactoring edits from a purposive sample of 118 Apache's merged PRs, hosted on GitHub. Our results reveal motivating factors behind refactoring-inducing PRs, technical aspects around the structure of review comments in refactoring-inducing and non-refactoring-inducing PRs, and guidelines for a more productive code reviewing settled on code enhancements through refactoring. Accordingly, we suggest directions for researchers, practitioners, tool builders, and educators to assist code review effectiveness in pull-based development.

Keywords Refactoring-inducing pull request · Modern code review · Empirical study

Communicated by: Ali Ouni

✉ Flávia Coelho
flaviacoelho@ufersa.edu.br

Nikolaos Tsantalis
nikolaos.tsantalis@concordia.ca

Tiago Massoni
massoni@computacao.ufcg.edu.br

Everton L. G. Alves
everton@computacao.ufcg.edu.br

1 Department of Computer Science, Federal Rural University of Semiarid, Mossoró, Brazil

2 Department of Computer Science, Concordia University, Montreal, Canada

3 Department of Computer Science, Federal University of Campina Grande, Campina Grande, Brazil

Published online: 31 October 2024

1 Introduction

In a code change-driven, tool-assisted, and asynchronous reviewing, named *Modern Code Review* (MCR), developers (*reviewers*) provide comments intending code improvements by manually examining the changes submitted by another developer (*author*) (Bacchelli and Bird 2013). Guided by change and collaboration, MCR is crucial in repository-based software as it occurs in *Agile development* and *continuous delivery* approaches, to efficiently produce deployable software releases (Agi 2001; Humble and Farley 2010). Git *Pull Requests* (PRs) accommodate a well-defined and collaborative reviewing (Git 2022), so fitting to the MRC practice. Accordingly, reviewers may suggest improvements to the code into independent working *branches* (development lines) before its *merging* (integration) to the main branch of a repository. For that, after forking a Git branch, a developer can implement changes through commits and open a PR to submit them for code reviewing. Chacon and Straub (2014). Collaborative development platforms based on Git have grown in the number of developers (over than 100 million at GitHub) around the world (Dohmke 2023).

Since changes may embrace new features, bug fixes, or other maintenance tasks, code enhancements may incorporate refactoring edits (Palomba et al. 2017). *Refactorings*, coined by Opdyke and Johnson (1990) and popularized by Fowler (1999), are performed to improve the design quality and maintainability of object-oriented software, whereas preserving its external behavior. In the MCR context, refactoring edits may emerge due to review comments and discussions around code quality issues and/or spontaneous actions of a PR developer (*author*) to refine her originally submitted code changes.

Empirical evidence points out the relevance of review comments and discussion for code refactoring at the PR level when studying merged PRs containing at least one refactoring (Pantiuchina et al. 2020). In a recent study that explored refactoring edits performed at subsequent commits (i.e., after the initial commits included in the PR when it is created) in merged PRs, we found a significant difference in the number of review comments and discussion length between refactoring-inducing and non-refactoring-inducing PRs, and a substantial number of refactoring edits induced by code review; thus identifying potential technical aspects that might shed light on the *refactoring-inducement* (Coelho et al. 2021).

In a refactoring-inducing PR (Definition 1), the refactoring edits emerge after the initial PR commit(s), resulting from review comments/discussion and/or spontaneous actions done by the PR author. Intending fair examinations, we consider the PRs that comprise at least one review comment.

Definition 1 A PR is refactoring-inducing if refactoring edits are performed in subsequent commits after the initial PR commit(s), as a result of the reviewing process or spontaneous improvements by the PR author. Let $U = \{u_1, u_2, ..., u_w\}$, a set of repositories in GitHub. Each repository u_q, $1 \leq q \leq w$, has a set of PRs $P(u_q) = \{p_1, p_2, ..., p_m\}$ over time. Each PR p_j, $1 \leq j \leq m$, has a set of commits $C(p_j) = \{c_1, c_2, ..., c_n\}$, in which $I(p_j)$ is the set of initial commits included in the PR when it is created and $S(p_j)$ is the set of subsequent commits incorporated into the PR after its creation, $I(p_j) \subseteq C(p_j)$ and $S(p_j) \subseteq C(p_j)$. A refactoring-inducing PR is that in which $\exists\ c_k \mid R(c_k) \neq \varnothing$ and $\exists\ c_i \mid RC(c_i) \neq \varnothing$, where $R(c_k)$ and $RC(c_i)$ respectively denote the set of refactoring edits performed in commit c_k and the set of review comments left in commit c_i, $|I(p_j)| < k \leq n$ and $c_i \in I(p_j) \cup S(p_j)$.

Figure 1 depicts a refactoring-inducing PR consisting of two initial commits ($c_1 - c_2$) and two subsequent commits ($c_3 - c_4$), which include refactoring edits, e.g., commit c_3 has one *Extract Method* edit. The PR also comprises review comments in commits c_2 and c_3. Our

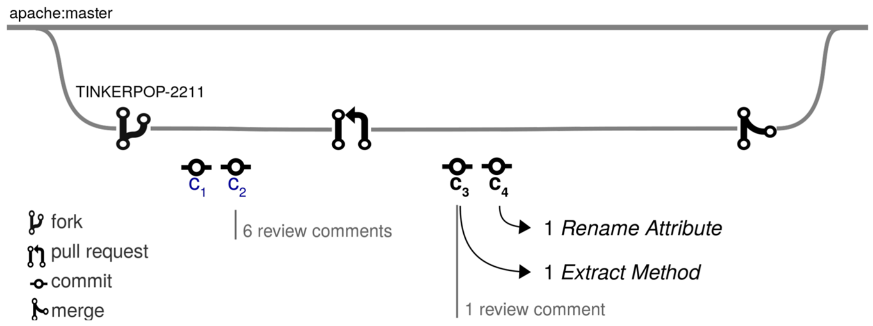


Fig. 1 A refactoring-inducing PR (Apache Tinkerpop #1110), illustrating initial commits ($c_1 - c_2$), subsequent commits ($c_3 - c_4$), and the presence of review comments

study explores refactoring edits in the subsequent PR commits ($c_3 - c_4$) and review comments/discussion in all PR commits ($c_1 - c_4$). To be concise, we use the format <Repository #number> to designate a specific repository's PR (e.g., Tinkerpop #1110).

Given that, we conjecture that refactoring-inducing and non-refactoring-inducing PRs have distinct characteristics regarding code review. We further scrutinize refactoring-inducing PRs with a qualitative study aiming to investigate reviewing-related aspects (review comments and discussion) to understand how code review can induce refactoring edits. For that, we search for patterns and particular characteristics intrinsic to those aspects in both refactoring-inducing and non-refactoring-inducing PRs.

1.1 Motivation

Review comments drive the majority of code changes in a PR (Beller et al. 2014), and consequently affect the time to merge a PR (Gousios et al. 2014). There is empirical evidence of an extensive review effort concerning changes containing refactoring edits, given that restructuring existing code may be complex (Wang et al. 2019; AlOmar et al. 2021). Despite works investigating the motivations behind refactoring (Paixão et al. 2020; Pantiuchina et al. 2020; Silva et al. 2016) and refactoring review criteria (AlOmar et al. 2022) at code review time, little is known about reviews that induce refactoring edits.

In a recent mixed-methods study, we explored technical aspects characterizing refactoring-inducing PRs (Coelho et al. 2021). For that, we randomly sampled 1,845 Apache's merged PRs, encompassing 4,702 subsequent commits, 6,556 detected refactorings, and 12,547 review comments, mined from 84 distinct repositories. Our quantitative analysis revealed 557/1,845 refactoring-inducing PRs (30.2%). By examining code churn (number of altered lines), discussion length, time to merge, and the number of subsequent commits, file changes, and review comments, we found significant differences between refactoring-inducing and non-refactoring-inducing PRs. In our qualitative analysis, we manually scrutinized a stratified sample of 228/557 refactoring-inducing PRs by cross-referencing 2,096 review comments to 1,891 detected refactorings. Code review induced refactoring edits in 133/228 refactoring-inducing PRs (58.3%). Our findings motivate further exploration of code reviewing-related aspects for a better knowledge of refactoring at the PR level.

Indeed, code review is a non-trivial task because of related technical and social aspects (Rigby et al. 2015). In 2013, Bacchelli and Bird manually examined review comments and determined motivations and challenges for MCR (Bacchelli and Bird 2013), thus leveraging actionable directions for researchers, practitioners, tool builders, and educators. Also, qualitative studies have discovered patterns in review comments that support a better understanding of MCR practice in the direction of *effectiveness* (acceptance of changes) and *efficiency* (time to accept changes) (Bosu et al. 2015; Li et al. 2017; Rahman et al. 2017; Ebert et al. 2021; Panichella and Zaugg 2020).

Through knowledge regarding the inherent characteristics of review comments in refactoring-inducing and non-refactoring-inducing PRs, we aim to advance the understanding of code review practices around pull-based development, and propose actionable improvements to this process. For instance, as discussions may affect the acceptance of PRs (Tsay et al. 2014), increase code design degradation (Uchôa et al. 2020), and influence the review time (Kononenko et al. 2018), it is relevant to scrutinize such an aspect in refactoring-inducing and non-refactoring-inducing PRs to recognize how they affect the refactoring practice.

Moreover, and perhaps mainly, we aim to enhance guidelines/good practices intending code review effectiveness and efficiency towards refactoring, thus supporting:

- the necessity for articulating appropriate review comments (Bosu et al. 2017; Ebert et al. 2021; Brown and Parnin 2020; Hossain et al. 2020); and
- the empirical evidence on the benefits of following best practices in code review regarding code quality improvements, knowledge transfer, team awareness, and sharing of code ownership – a practical challenge because code review embraces technical and social aspects (Chouchen et al. 2021; MacLeod et al. 2018).

1.2 Research Overview

It is noteworthy that we consider code review practices at the PR level aiming to explore a well-defined development scope, thus scrutinizing review comments from the initial commits to subsequent commits of a PR. Accordingly, we could gain a global comprehension of contributions to the original code through both commits and reviewing-related aspects. Empirical evidence that PR discussions lead to additional refactoring edits (Pantiuchina et al. 2020) inspired such a conception.

We scrutinized a *purposive sample* of 118 merged PRs, 923 review comments, 361 refactoring edits, and the experience of their reviewers. We selected those 118 PRs from 1,639 Apache's merged PRs, previously mined from GitHub. This setting can provide a well-defined structure (of contributions and contributors) to understand motivations, peculiarities, and code review practices as an initial ground for investigations regarding refactoring-inducing PRs. To establish this scenario, we considered the empirical insights on the contexts and organizational structures as influencing factors for code reviewing provided by Baysal et al. (2016).

Our qualitative study encompasses four rounds of analysis. At each one, we investigated review comments, discussion, and the reviewers' experience from the purposive sample of PRs while cross-referencing their detected refactoring edits. Our results reveal motivating factors behind refactoring-inducing PRs, technical aspects of the structure of review comments in both refactoring-inducing and non-refactoring-inducing PRs, and guidelines for a more productive code review towards code refactoring.

1.3 Contributions

We point out the main contributions of this work:

- To the best of our knowledge, this is the first pure qualitative study investigating code reviewing-related aspects in the context of refactoring-inducing PRs (Definition 1).

- This work analyses code review in real-life and collaborative repositories driven by pull-based development.
- Our purposive sample comprises only merged PRs that did not suffer rebasing and their respective set of validated refactoring edits in order to mitigate threats to validity.
- We make available a complete reproduction kit, including the datasets, implemented scripts, worksheets, and instructions to enable replications and future research (Coelho 2024).

2 Motivating Study

In our previous study (Coelho et al. 2021), we discovered significant differences between refactoring-inducing and non-refactoring-inducing PRs in terms of code churn, number of subsequent commits, number of file changes, number of review comments, discussion length, and time to merge. Moreover, we found refactoring edits induced by code review in almost 60% of refactoring-inducing PRs in a stratified sample (number of refactoring edits as criteria). Given that, we speculate that there are distinctions between refactoring-inducing and non-refactoring-inducing PRs, in terms of the structure of review comments. In specific, we hypothesize that review comments that induce refactoring edits may follow some particular structuring pattern. Since code review and refactoring are time-consuming tasks (Alves et al. 2018; Gousios et al. 2014; Szőke et al. 2014), a better understanding of review practices in refactoring-inducing PRs might provide directions, such as guidelines, in order to improve the practitioners' effectiveness and productivity (e.g., reducing delays in merging code changes in pull-based development) towards code refactoring.

As a preliminary study, we analyzed the time to merge in a sample of 65 refactoring-inducing PRs, in turn, classified into (A) refactoring-inducing PRs in which code review induced refactorings (49/65) and (B) refactoring-inducing PRs in which only authors proposed refactorings (16/65). We observed that PRs in category A take 8.8 on average (SD = 13.8) and three on median (IQR = 10) days to merge, while the others (category B) take 24.8 on average (SD = 44.9) and 5.5 on median (IQR = 20.5) days (Coelho 2022). Therefore, merging a PR in Category B takes three times as long as in Category A. Also, by manually examining a few PRs, we discovered that uncertain review comments, which incorporate expressions like "*wondering if...*", usually have no effect (e.g., Flink #9143). However, when structured as direct sentences, review comments often induce refactoring edits (e.g., "*is an array list here a little bit over-kill?*" caused a *Change Type* refactoring in Dubbo #3299). This result provides insights into the pertinence of an in-depth examination of review comments aiming to identify patterns that, in turn, may assist us in designing guidelines for a more effective and efficient code review in pull-based development.

3 Methodology

Our qualitative study investigates code reviewing-related aspects (review comments, discussion, and reviewers' experience) intending to characterize code review in refactoring-inducing PRs, guided by the following **research questions**:

RQ$_1$: How are review comments characterized in refactoring-inducing and non-refactoring-inducing PRs? We first identified the characteristics of review comments in both groups.

RQ$_2$: What are the differences between refactoring-inducing and non-refactoring-inducing PRs, in terms of review comments? Then, we investigated the existence of similarities/dissimilarities regarding review comments in both groups by contrasting their corresponding characteristics.

RQ$_3$: How do reviewers suggest refactorings in refactoring-inducing PRs? We scrutinized the review comments in refactoring-inducing PRs to identify patterns for suggesting refactorings.

RQ$_4$: Do suggestions of refactoring justify the reasons? While exploring the refactoring suggestions, we explored whether/how reviewers provide rationales for their proposals.

RQ$_5$: What is the relationship between suggestions and actual refactorings in refactoring-inducing PRs? While studying the suggestions of refactoring in review comments, we also investigated patterns that would denote any relationship between suggestions and actual refactorings.

It is worth clarifying that, in a previous mixed-methods work, we discovered differences between refactoring-inducing and non-refactoring-inducing PRs by investigating quantitative aspects of Apache's merged PR (number of file changes and subsequent commits, code churn, and time to merge) and code review (number of review comments and discussion). Moreover, our qualitative analysis revealed that code review induced refactoring edits in almost 60% of a stratified sample of refactoring-inducing PRs (Coelho et al. 2021). Based on these results, this study is a second step aiming to gain an in-depth understanding of the similarities/dissimilarities between refactoring-inducing and non-refactoring-inducing PRs regarding code reviewing-related aspects. Accordingly, driven by a qualitative approach, we concentrate on identifying the inherent characteristics of review comments in both categories (RQ$_1$) and recognizing their differences (RQ$_2$). Then, we explore in more depth refactoring-inducing PRs to know how review comments suggest refactoring edits (RQ$_3$), whether the refactoring suggestions provide reasons (RQ$_4$), and the relation between these suggestions and the actual refactoring edits (RQ$_5$). Note that these research questions follow a chain of evidence ranging from RQ$_1$–RQ$_2$ to RQ$_3$–RQ$_5$ to better grasp the code review practice in the presence of refactoring edits.

To address our research questions, we considered merged PRs from Apache's repositories on GitHub. We collected only merged PRs because they reveal finalized actions; thus, we could explore the review comments/discussion and the refactoring edits performed that constitute the actual history of PRs. Note that discussion, as we deemed, denotes either a dialog between a PR author and reviewer(s) or an author's response in opposition to one or more review comments. We gathered repositories from the Apache ecosystem due to its popularity and practical relevance of contributions in the *Open-Source Software* (OSS) scenario (more than 8,200 contributors and 350 projects, in turn, completely migrated to GitHub since February 2019) (Apa 2019a), so denoting a high maturity in pull-based development. In particular, we picked only Java repositories given almost 28% of Apache's source code is developed in Java language in contrast to other languages, such as C++ (7.3%) and Python (1.5%) (Apa 2020).

To mine refactoring edits in those repositories, we chose the RefactoringMiner – a state-of-the-art refactoring detection tool (precision of 97.96% and recall of 87.2%), which has been shown superior to competitive tools to detect refactoring edits applied to Java source code that follow pull-based development (Tsantalis et al. 2018). We considered RefactoringMiner version 2.0 (Tsantalis et al. 2020), because it supports up to 40 different types of refactoring, including low-level, medium-level, and high-level refactorings, allowing us to consider a comprehensive and well-defined list of refactorings. Such a version may detect edits of refactoring well-known by developers, including a few ones cataloged by Fowler (2018): *Extract Method*, *Inline Method*, *Rename Method*, *Move Method*, *Move Attribute*, *Rename Class*, *Extract and Move Method*, *Rename Package*, *Extract Variable*, *Inline Variable*, *Parameterize Variable*, *Rename Variable*, *Rename Parameter*, *Rename Attribute*, *Replace Variable with Attribute*, *Replace Attribute (with Attribute)*, *Merge Variable*, *Merge Parameter*, *Merge Attribute*, *Split Variable*, *Split Parameter*, *Split Attribute*, *Change Variable Type*, *Change Parameter Type*, *Change Return Type*, *Change Attribute Type*, *Extract Attribute*, *Move and Rename Method*, *Move and Inline Method*, *Pull Up Method*, *Pull Up Attribute*, *Push Down Method*, *Push Down Attribute*, *Extract Superclass*, *Extract Interface*, *Move Class*, *Move and Rename Class*, *Extract Class*, *Extract Subclass*, and *Move and Rename Attribute* (Ref 2018). Note that newer RefactoringMiner versions might identify additional refactoring types; however, we speculate that this fact does not change our qualitative findings. Notwithstanding, those extra refactoring types may be the focus of further research.

Our qualitative study considers code review comments (*code review dataset*) and refactoring edits (*refactorings dataset*) previously detected in 1,639 merged PRs. We provide details about the complete mining process of the merged PRs and their refactoring edits in Coelho (2022); the datasets are publicly available (Coelho 2024). We manually inspected those PRs to deal with *rebasing* issues (some PR commits may be due to external changes – outside the scope of the code review sequence, conveying a threat to the validity (Paixão and Maia 2019)), thus providing a validated dataset of merged PRs for further investigations. A detailed explanation of all different PR merge options is available in our previous work (Coelho et al. 2021).

As a whole, at each round of our qualitative study, we examined review comments, discussion, and reviewers' experience of a purposive sample of PRs fitting a valuable scenario to the current purposes of the analysis, while cross-referencing their detected refactorings (from *refactorings dataset*). We adopted such non-probability sampling to select refactoring-inducing and non-refactoring-inducing PRs from *code review dataset*. We followed that sampling strategy until we reached *data saturation* (when no new information emerges), in turn, was achieved after four rounds of analysis. To address our research questions, each round of analysis follows the procedures depicted in Fig. 2.

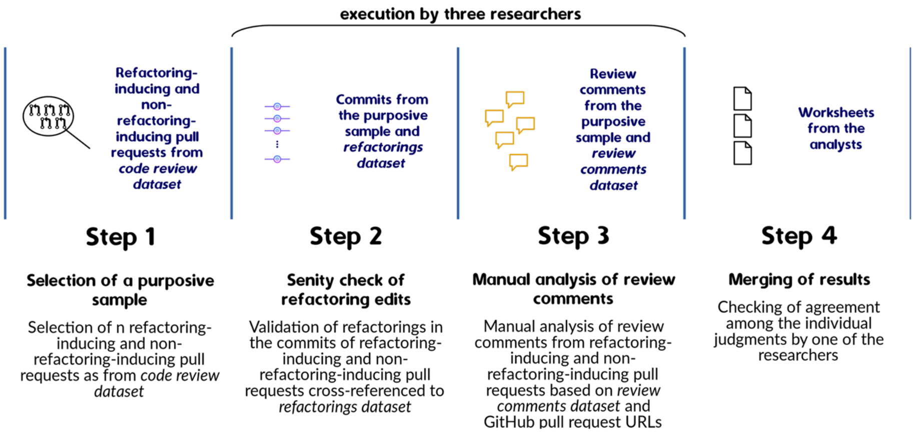

execution by three researchers
Refactoring-inducing and non-refactoring-inducing pull requests from *code review dataset*
Commits from the purposive sample and *refactorings dataset*
Review comments from the purposive sample and *review comments dataset*
Worksheets from the analysts
Step 1
Selection of a purposive sample
Selection of n refactoring-inducing and non-refactoring-inducing pull requests as from *code review dataset*
Step 2
Senity check of refactoring edits
Validation of refactorings in the commits of refactoring-inducing and non-refactoring-inducing pull requests cross-referenced to *refactorings dataset*
Step 3
Manual analysis of review comments
Manual analysis of review comments from refactoring-inducing and non-refactoring-inducing pull requests based on *review comments dataset* and GitHub pull request URLs
Step 4
Merging of results
Checking of agreement among the individual judgments by one of the researchers

Fig. 2 An overview of round *i* of our qualitative study

We clarify that our study design does not encompass a survey among the PR reviewers to validate our findings because this approach would require recent (open) PRs so that the involved reviewers have fresh memory discussions. Our dataset is relatively old with PRs ranging from 2014–2019. Contacting the code reviewers on such old PRs would not be effective, as they would hardly remember their thoughts, and thus not willing to discuss these PRs with researchers. Even the best-designed studies following the *firehouse interview method* (Silva et al. 2016), in which the researchers contacted the developers within 24 hours after their actions, achieved a response rate of 42%.

Next, we describe how we deal with the review comments and discussions (Sections 3.1 to 3.4) and the reviewers' experience (Section 3.5).

3.1 Selection of a Purposive Sample

We chose purposive sampling because it supports an in-depth understanding of the data by exploring scenarios suitable at each round, in line with emergent patterns or ideas (Patton 2014). Note that, in all rounds, we selected the most representative samples, in line with emergent patterns, intending to obtain more accurate results than those achieved by selecting other probability sampling strategies. For instance, in the second round, we explored refactoring-inducing PRs containing only one low-level refactoring edit. Then, in the third round, we examined refactoring-inducing PRs comprising high-level refactoring edits in order to investigate whether the emerged patterns from Round 2 remain in their presence. Table 1 outlines the main objective of each round of analysis. The emerged patterns from Round 1 continue until Round 4, in which we reach a saturation point.

We empirically considered 20 as the minimum size for the purposive samples by following Creswell's guidelines (Creswell 2012) on the number of interviewers during a grounded theory study, which comes close to our characterization study (when investigating comments from reviewers), aiming at developing a well-saturated theory. Thus, as a PR has at least one reviewer, we took into account 20: ten refactoring-inducing PRs and ten non-refactoring-inducing PRs. Accordingly, we established 20 as a baseline in Rounds 1 and 2. Then, we employed theoretical saturation to determine the final sample size in Rounds 3 and 4; in particular, we deemed unexplored settings of code refactoring as a general rule to achieve a saturation point. Next, we detail how we built the samples; note that we apply a random purposeful strategy in all rounds.

For the first round of analysis (sample 1), looking for a fair comparison between groups when addressing RQ_1 and RQ_2, we randomly selected ten refactoring-inducing and ten non-refactoring-inducing PRs that contain five review comments and two reviewers – intermediate

Table 1 Summary of the objective of each round of the qualitative analysis

Round	Main objective
1	Investigating an initial random sample of PRs
2	Checking if emerged patterns remain in refactoring-inducing PRs consisting of only one refactoring
3	Exploring if emerged patterns persist in refactoring-inducing PRs consisting of high-level refactorings
4	Inspecting if emerged patterns continue regardless of the sequence of refactoring edits

values regarding the average and median of the number of review comments and reviewers in both groups (in the 1,639 PRs).

We also considered 20 as the size for the second purposive sample (sample 2): ten refactoring-inducing and ten non-refactoring-inducing PRs that contain only one subsequent commit. We investigated whether the patterns that emerged from distinct compositions of refactoring edits (Round 1, in which the number of refactorings is 6.6 on average and 3.5 on the median) even remain in refactoring-inducing PRs consisting of only one refactoring edit – the most simple setting of refactoring in a refactoring-inducing PR. We addressed PRs comprising a single subsequent commit to compare refactoring-inducing and non-refactoring-inducing PRs (RQ_1 and RQ_2).

In the third purposive sample (sample 3, Round 3), to address RQ_3 to RQ_5, we considered 13 refactoring-inducing PRs from 36 that present high-level refactorings to explore whether the emergent patterns from the previous rounds (which comprise less complex PRs) persist. Those 13 refactoring-inducing PRs embrace a diversified setting of refactoring edits of distinct types (found in those 36), including only one type of refactoring (e.g., Dubbo #3654) and a mix of types of refactoring (e.g., Incubator-Iceberg #183). We decided to explore refactoring-inducing PRs consisting of high-level refactorings because only 2/20 (10%) refactoring-inducing PRs contain high-level refactoring edits in the previous samples (Rounds 1 and 2): Flink #7945 and Kafka #5194. To deal with RQ_1 and RQ_2, we examined a randomly selected sample of 13 non-refactoring-inducing PRs that present ten review comments – a median of the number of review comments in that 13 refactoring-inducing PRs.

In the fourth purposive sample (sample 4, Round 4), to address RQ_3 to RQ_5, we selected the 26 refactoring-inducing PRs that present distinct sequences of refactoring edits (of different types) in the PR commits history (e.g., edits of *Rename Variable* and *Extract Variable* in a single commit against ones in two separated commits). In particular, we explored whether the emergent patterns from the previous rounds persist regardless of the sequence of refactoring edits applied to the commits history. To address RQ_1 and RQ_2, we investigated a randomly selected sample of 26 non-refactoring-inducing PRs that present seven review comments – the median value of the number of review comments in that 26 refactoring-inducing PRs.

Table 2 summarizes the number of refactoring-inducing and non-refactoring-inducing PRs, review comments, subsequent commits, and refactoring edits considered in each round of analysis. Table 3 displays a few statistical summaries regarding code reviewing-related aspects and time to merge for the PRs. The different number of refactoring-inducing and non-refactoring-inducing PRs in Round 4 is due to the manual validation of refactorings mined by RefactoringMiner, as explained in Section 3.2. The average and the median number of refactoring edits consisted of 6.6 (SD = 7.6) and 3.5 (IQR = 6.2) in Round 1; 1.0 (SD = 0.0)

Table 2 Summary of the purposive samples

Round	Number of refactoring-inducing PRs	Number of non-refactoring-inducing PRs	Number of review comments	Number of subsequent commits	Number of refactoring edits
1	10	10	100	40	66
2	10	10	87	20	10
3	13	13	327	126	208
4	27	25	409	160	77
Total	*60*	*58*	*923*	*346*	*361*

Table 3 Statistical summary (code review aspects and time to merge) of the purposive samples

Round	Number of reviewers	Number of review comments	Time to merge (in number of days)
Refactoring-inducing PRs			
1	2.0 (0.0)	5.0 (0.0)	3.4 (6.7)
	2.0 (0.0)	5.0 (0.0)	1.0 (2.2)
2	1.7 (0.5)	4.1 (4.6)	3.9 (6.8)
	2.0 (0.7)	2.0 (2.2)	1.0 (2.7)
3	3.1 (1.1)	15.1 (13.7)	11.5 (14.8)
	3.0 (1.0)	10.0 (14.0)	6.0 (8.0)
4	2.3 (0.9)	8.7 (5.7)	21.9 (36.4)
	2.0 (1.0)	7.0 (8.5)	7.0 (15.5)
Non-refactoring-inducing PRs			
1	2.0 (0.0)	5.0 (0.0)	2.5 (3.6)
	2.0 (0.0)	5.0 (0.0)	1.0 (2.7)
2	2.2 (0.6)	4.6 (4.0)	2.6 (2.3)
	2.0 (0.7)	3.5 (3.2)	2.5 (4.5)
3	2.7 (0.9)	10.0 (0.0)	10.9 (23.2)
	3.0 (1.0)	10.0 (0.0)	2.0 (7.0)
4	2.4 (0.7)	7.0 (0.0)	6.7 (8.2)
	2.0 (1.0)	7.0 (0.0)	3.0 (11.0)

where <number> (<number>) denotes the average (standard deviation) and median (interquartile range)

and 1.0 (IQR = 0.0) in Round 2; 16.0 (SD = 18.6) and 10.0 (IQR = 12.0) in Round 3; and 2.8 (SD = 1.1) and 2.0 (IQR = 2.0). The magnitude of refactoring-inducing and non-refactoring-inducing PRs increases in the following order (Table 4): *sample 2 < sample 1 < sample 4 < sample 3* when considering size-related aspects (number of subsequent commits, number of file changes, and code churn).

Table 4 Statistical summary (magnitude) of the purposive samples

Round	Number of subsequent commits	Number of file changes	Number of added lines	Number of deleted lines
Refactoring-inducing PRs				
1	2.3 (1.2)	9.1 (9.3)	103.9 (185.7)	50.5 (61.7)
	2.0 (1.7)	5.5 (12.5)	31.0 (47.0)	23.0 (43.7)
2	1.0 (0.0)	1.7 (0.7)	38.5 (66.1)	28.7 (59.1)
	1.0 (0.0)	2.0 (1.0)	6.0 (31.7)	5.0 (16.2)
3	5.8 (3.6)	26.5 (34.6)	550.0 (1,198.8)	297.9 (490.7)
	5.0 (6.0)	16.0 (22.0)	143.0 (332.0)	104.0 (281.0)
4	3.8 (3.3)	9.1 (12.9)	278.9 (797.6)	54.6 (79.3)
	3.0 (2.5)	5.0 (6.0)	39.0 (99.5)	23.0 (27.5)

Table 4 continued

Round	Number of subsequent commits	Number of file changes	Number of added lines	Number of deleted lines
Non-refactoring-inducing PRs				
1	1.7 (0.9)	3.0 (1.9)	21.8 (25.3)	14.8 (17.5)
	1.5 (1.0)	2.0 (2.5)	8.5 (36.0)	7.5 (16.0)
2	1.0 (0.0)	2.1 (1.7)	104.1 (278.5)	97.7 (280.2)
	1.0 (0.0)	1.5 (1.0)	3.5 (39.7)	4.0 (7.7)
3	3.8 (1.9)	7.7 (4.6)	47.2 (48.6)	29.9 (39.4)
	3.0 (2.0)	6.0 (4.0)	23.0 (40.0)	18.0 (16.0)
4	2.2 (2.2)	4.4 (4.9)	54.2 (92.4)	31.3 (63.3)
	2.0 (2.0)	3.0 (4.0)	18.0 (21.0)	6.0 (23.0)

where <number> (<number>) denotes the average (standard deviation) and median (interquartile range)

Furthermore, Table 5 describes the Apache repositories of the PRs under exploration, accompanied by their frequency in refactoring-inducing and non-refactoring-inducing PRs. As a whole, we investigated PRs from 27 distinct repositories, in which Kafka, Dubbo, and Beam are more frequent, totalizing 24/118 (20.34%), 15/118 (12.71%), and 12/118 (10.17%), respectively.

Table 5 Repositories of the purposive samples

Repository	Purpose	RI	nonRI
Accumulo	Sorted, distributed key/value store	1/60	2/58
Avro	Data serialization system	1/60	0/58
Beam	Batch and streaming data processing	4/60	8/58
Brooklyn-Server	Managing cloud applications	2/60	1/58
Cloudstack	Deploying and managing networks of virtual machines	3/60	6/58
Commons-Text	Library for handling strings	0/60	1/58
Dubbo	Web and RPC framework	7/60	8/58
Flink	Framework and distributed processing engine	7/60	4/58
Fluo	Distributed processing system	1/60	1/58
Hadoop	Framework for distributed processing	0/60	1/58
Incubator-Iceberg	Open table format for analytic datasets	2/60	1/58
Incubator-Iotdb	IoT native database	0/60	2/58
Incubator-Pinot	Real-time analytics open source platform	1/60	1/58
Kafka	Distributed event streaming platform	14/60	10/58
Knox	REST API and application gateway	2/60	0/58
Logging-Log4j	Logging framework	1/60	0/58
Parquet-Format	Data file format for storage and retrieval	0/60	1/58
Plc4x	Set of libraries for interacting with PLCs	0/60	1/58

Table 5 continued

Repository	Purpose	RI	nonRI
Rocketmq-Externals	Real-time data processing platform	1/60	0/58
Samza	Distributed stream processing framework	2/60	0/58
Servicecomb-Java-Chassis	Java SDK to build microservices	2/60	4/58
Sling-Org-Apache-Sling-Feature-Analyser	Framework for RESTful Web applications	1/60	0/58
Struts	Framework for Java Web applications	1/60	1/58
Tika	Detection and extraction of metadata	1/60	0/58
Tinkerpop	Graph computing framework	2/60	4/58
Tomee	JakartaEE application server	3/60	1/58
Usergrid	Backend-as-a-service	1/60	0/58

where RI and non-RI denote the frequency of each repository in refactoring-inducing and non-refactoring-inducing PRs, respectively, in our purposive sample

3.2 Sanity Check of Refactoring Edits

As shown in Fig. 2, at each round, three researchers individually checked all PR commits, searching for refactoring edits (Step 2), by exploring the type and description of them as stored in the *refactorings dataset*. For that, after selecting each purposive sample (Round 1 – Round 4), they individually examined the changed code snippets across the PR subsequent commits at GitHub, by cross-referencing them with the refactoring edit(s) as stored in the *refactorings dataset* aiming to identify true negatives, false positives, and false positives. To document their observations, each researcher filled out the fields *confirmed_refactoring_flag* and *notes* in a worksheet (detailed in Section 3.3). Then, in a remote meeting, they collectively checked their observations. With no disagreements or disputes, they identified that RefactoringMiner mistakenly detected an edit as a *Rename Method* (Kafka #6565). It also did not identify an *Extract Attribute* (Accumulo-Examples #19), an *Extract Variable* (Tinkerpop #893), and a *Rename Method* (Kafka #7132). RefactoringMiner achieved a precision of 99.7% and recall of 99.2% in the refactoring detection in 118 PRs (Table 6).

3.3 Manual Analysis of Review Comments

Each researcher examined the review comments from the PRs in each purposive sample (Step 3), assisted by the *refactorings dataset*. To clarify, besides seeking patterns in review

Table 6 Results of the manual validation of refactoring edits mined by RefactoringMiner

Sample	Number of true positives	Number of false positives	Number of false positives	Precision (%)	Recall (%)
1	66	0	0	100.0	100.0
2	10	1	1	90.9	90.9
3	208	0	0	100.0	100.0
4	77	0	2	100.0	97.5
Total	*361*	*1*	*3*	*99.72*	*99.18*

comments, the researchers cross-referenced the type of refactorings performed (if so) and the review comments left in all commits of a PR. In such a procedure, it was crucial to directly examine PRs at GitHub because we could access commits and review comments in chronological order.

For each sample, the researchers filled a worksheet structured with the following fields:

- *repo*: repository name,
- *pr_number*: PR number,
- *category*: refactoring-inducing PR or non-refactoring-inducing PR,
- *pr_url*: PR URL,
- *commit*: commit SHA,
- *initial_flag*: whether a commit is an initial or subsequent commit,
- *refactoring_type*: type of a refactoring edit,
- *refactoring_detail*: description of a refactoring edit, as RefactoringMiner output,
- *confirmed_refactoring_flag*: if a refactoring edit is a true positive,
- *covered_refactoring_flag*: if a refactoring edit was induced by code review,
- *floss_refactoring_flag*: if there is the presence of floss refactoring in a commit,
- *direct_review_comment_flag*: if a review comment directly suggests a refactoring edit,
- *discussion_flag*: if there was discussion related to a review comment in a commit,
- *rationale_flag*: if a review comment presents a rationale to suggest a refactoring edit, and
- *notes*: specific comments of a researcher.

In this subjective analysis, the researchers considered the refactoring inducement in settings where review comments, either explicitly suggested refactoring edits or left any actionable recommendation that induced refactoring, to answer the research questions. For example, "*... the name is really misleading ...*" induced a *Rename Method* (Avro #525), whereas "*Won't you need to use a single instance for both arguments?*" inspired an *Extract Attribute* (Beam #4407). It is noteworthy that the researchers have a background in such an analysis, given their experience in our previous study (Coelho et al. 2021).

3.4 Merging of Judgements

In a remote meeting, at the end of each round, one researcher analyzed all individual judgments while filling out a *synthesizing worksheet* for the sample. As a decision criterion, we considered the agreement of responses by at least two researchers when analyzing the fields of the worksheets. Leveraging the researchers' expertise in subjective data analysis concerning code review and refactoring induction, the decision criterion was adequately fulfilled (100% of the samples attained at least two agreements, with no dispute cases). Disagreement cases (e.g., regarding a missed indirect refactoring suggestion) were rare and promptly treated by the researchers at the remote meetings. This setting occurred about twice in each purposive sample.

Driven by emerging patterns, the same researcher explored the field 'notes' in the synthesizing worksheet in order to answer the research questions. Next, the three researchers engaged in meetings to discuss those answers, aiming to achieve a concluding comprehension concerning the sample (Step 4). For that, they carried out the following procedures to deal with each research question: (1) read the proposed answer to the research question, (2) highlight the emergent patterns, (3) confirm/refute/include the emergent patterns, and (4) revise the answer.

It should be noted that the answers to the research questions are intrinsically subjective and transcend the worksheets' fields. To clarify, to understand the "how" (RQ_1 and RQ_3),

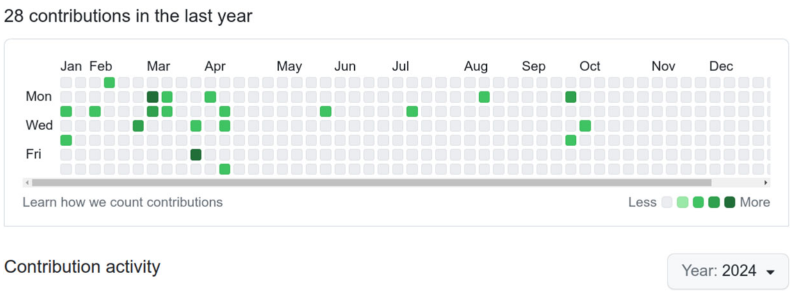

28 contributions in the last year
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
Mon
Wed
Fri
Learn how we count contributions
Less More
Contribution activity
Year: 2024

Fig. 3 Example of a GitHub profile (author of Kafka #5784)

we searched for patterns that, in turn, rely on a discerning interpretation by the researchers. The "what"(RQ_2 and RQ_5) also requires some comparative judgment in light of the identified patterns. RQ_4 is less reliant on patterns but still deserves attention, as refactoring suggestions may vary in structure and content (e.g., straightforward versus complex review comments). Therefore, we made an effort to carefully consider group brainstorming sessions, discussions, and revisiting PRs on GitHub as needed, aiming to build a comprehensive and collective understanding of each sample. Accordingly, the concluding analysis of each round denotes incremental knowledge on refactoring-inducing PRs. We document those intermediate judgments and respective emerged patterns in our reproduction kit (Coelho 2024).

3.5 Reviewers' Experience

Apache Foundation designates roles for contributors (Apa 2021). A *contributor* is a developer who contributes to a project in the form of code or documentation; a *committer* is a contributor who has write access to the code repository, and a *Project Management Committee* (PMC) *member* is a committer who has write access to the code repository and can approve/disapprove the changes. Not all PR participants indicate their Apache roles. Thus, we inferred such a scenario from examining the profile of authors and reviewers. In addition to Apache contributors, we recognized that committers and PMC members also submit PRs to Apache repositories. It is noteworthy that we did not include the PRs with no review in Java code to compute the experience.

We considered the number of contributions of the Apache contributors (PR authors and reviewers) as a proxy for their experience. For that, we computed the number of contributions taking into account the joining date of their profiles in GitHub until the PR creation date under investigation. Namely, we summed all the number of contributions by year, in the range, for each developer – a manual and time-consuming process because we needed to check each profile in line with the associated PR creation date. Therefore, a developer might have distinct computed experience in different PRs. To illustrate our counting strategy, consider Kafka #5784, created in October 2018. Each one of the contributors has a profile from which we can access the number and the description of their contributions by year[1], as exemplified in Fig. 3. The PR author joined GitHub in 2015; thus, we manually counted the number of his contributions from 2015 until September 2018 (the previous month to the PR date creation), figuring 1,123 contributions. Correspondingly, we computed the number of contributions for all PR authors and reviewers in our sample.

[1] https://github.com/rajinisivaram

We explored such a subject because there is empirical evidence that the reviewer experience is the main factor influencing code review quality (Kononenko et al. 2015, 2016; Thongtanunam et al. 2016; Bosu et al. 2017; Rahman et al. 2017). Nevertheless, it is not feasible to accurately compute reviewer experience (Kononenko et al. 2018). For instance, Rigby et al. figured it in terms of the time a developer has been with a project whereas Baysal et al. calculated the number of previous reviews and classified them in line with the reviewing efforts (Baysal et al. 2016). We consider the number of contributions as a proxy for reviewer experience, as using qualitative methods for assessing reviewer experience would be very time-consuming, given the large number (thousands) of code reviews in our sample. Note that whereas Rigby et al. found that reviewers typically have more experience than authors when investigating Apache HTTP Server, we extend the knowledge of the experience of Apache developers by studying their experience in pull-based development (on refactoring-inducing and non-refactoring-inducing PRs).

We applied statistical hypothesis testing intending to analyze the computed experience of PR authors and reviewers, driven by a comparison between refactoring-inducing and non-refactoring-inducing PRs. Accordingly, these are the hypotheses:

H_{1_0}: There is no difference between authors' experience in refactoring-inducing and non-refactoring-inducing PRs.
H_{1_a}: Authors of refactoring-inducing PRs are less experienced than authors of non-refactoring-inducing PRs.
H_{2_0}: There is no difference between reviewers' experience in refactoring-inducing and non-refactoring-inducing PRs.
H_{2_a}: Reviewers of refactoring-inducing PRs are more experienced than reviewers of non-refactoring-inducing PRs.
H_{3_0}: There is no difference between the experience of authors and reviewers in refactoring-inducing PRs.
H_{3_a}: Authors are less experienced than reviewers in refactoring-inducing PRs.
H_{4_0}: There is no difference between the experience of authors and reviewers in non-refactoring-inducing PRs.
H_{4_a}: Authors are more experienced than reviewers in non-refactoring-inducing PRs.

For that, we executed each hypothesis testing in line with the following workflow, guided by Burdess (2010):

1. Definition of null and alternative hypotheses.
2. Checking of the assumptions for parametric statistical tests: data normality by using the *Shapiro-Wilk* test and homogeneity of variances via *Levene's* test. The independence assumption is already met (i.e., a developer is either an author or a reviewer in a PR).
3. Built on a significance level of 5%, performing of either parametric independent *t-test* and *Cohen's d*, or non-parametric *Mann Whitney U* test, in line with the output from 2.

Note that, since we examine differences in a specific direction in each hypothesis, we run one-tailed tests.

4 Results and Discussion

This qualitative study aims to advance the knowledge regarding code review in refactoring-inducing PRs; therefore, its findings are supplementary to ones from our previous study

(Coelho et al. 2021). For instance, note that we designed distinct sampling strategies in the two studies – a stratified sampling to explore refactorings induced by code review against a purposive sampling in this study.

We organize this section as follows. Section 4.1 describes a few intrinsic characteristics of our dataset, which are crucial to contextualize how we address the research questions in light of our study objective. Sections 4.2 to 4.6 answer the research questions. In Section 4.7, we discuss how our findings enhance the knowledge regarding code review in refactoring-inducing PRs.

4.1 Preliminary Results

4.1.1 Refactoring-inducement Rates

Table 7 specifies the refactoring-inducing PRs in each sample. They are grouped because we found refactoring-inducing PRs in which (i) code review induced all refactoring edits, (ii) the

Table 7 Refactoring-inducement rates

Sample	Result	Refactoring-inducing PRs
with refactorings induced exclusively by code review		
1	4/10 (40.0%)	Dubbo #3299, Flink #9143, Fluo #837, Kafka #5194
2	8/10 (80.0%)	Beam #4407 #4458, Brooklyn-Server #1049, Kafka #5784 #7132, Samza #1051, Servicecomb-Java-Chassis #346, Tomee #275
3	4/13 (30.8%)	Cloudstack #2071 #3454, Kafka #4757 #5590
4	14/27 (51.8%)	Brooklyn-Server #964, Cloudstack #2833, Dubbo #3174 #3257, Flink #8620, Kafka #4796 #6853, Knox #69 #74, Sling-Org-Apache-Sling-Feature-Analyser #16, Struts #43, Tika #234, Tinkerpop #1110, Tomee #407
Total	*30/60 (50.0%)*	
with refactorings led exclusively by the author		
1	3/10 (30.0%)	Beam #4460, Flink #7971, Samza #1030
2	2/10 (20.0%)	Incubator-Pinot #479, Kafka #5423
3	4/13 (30.8%)	Beam #6261 Dubbo #3654, Kafka #6657, Usegrid #102
4	7/27 (25.9%)	Accumulo #151, Dubbo #2445 #4099, Logging-Log4j #213, Kafka #4574, Tinkerpop #893, Tomee #89
Total	*16/60 (26.7%)*	
with refactorings induced by code review and other ones led by the author		
1	3/10 (30.0%)	Dubbo #2279, Flink #7945 #7970
2	0/10 (0.0%)	None
3	5/13 (38.4%)	Flink #8222, Incubator-Iceberg #119 #183, Kafka #4735, Servicecomb-Java-Chassis #678
4	6/27 (22.3%)	Avro #525, Flink #7165, Kafka #5501 #5946 #6848, Rocketmq-Externals #45
Total	*14/60(23.3%)*	

PRs' authors led all refactorings, and (iii) both code review and authors triggered refactoring edits, that is, both (i) and (ii) occurring simultaneously.

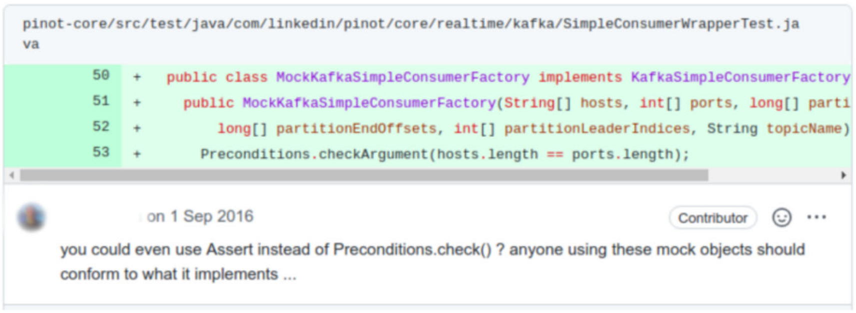
Authors led all refactoring edits in 16/60 (26.7%) of refactoring-inducing PRs; to clarify, a refactoring-inducing PR in which a refactoring was entirely led by its author expresses that there is no mention/suggestion (direct or indirect) of the refactoring in the review comments. In this scenario, Fig. 4 displays a single review comment in Incubator-Pinot #479, which had no effect in terms of refactoring. Even so, the author led an *Extract Variable*.

We detected 30/60 (50.0%) refactoring-inducing PRs that incorporate refactorings induced exclusively by code review – a number close to the result reported in our previous work (58.3%) (Coelho et al. 2021). In this case, review comments induced refactoring edits either by explicitly suggesting refactorings (e.g., *"How about renaming to ...?"* in Samza #1051) or leaving any actionable proposal that induced refactoring (e.g., *"New private method with obfuscated name should have some comment describing what it's supposed to do"* inspired a *Rename Method* edit in Fluo #837). As well, Fig. 5 shows a review comment indicating an outdated protocol in Cloudstack #2071, which influenced a *Rename Package* edit.

We also observed 14/60 (23.3%) refactoring-inducing PRs present refactoring edits induced by code review and others led by the PR authors. For example, in Flink #7945, the author self-performed refactoring edits in the first and second subsequent commits. In contrast, he carried out refactorings in the third subsequent commit as suggested in review comments.

In particular, by covering groups (i) and (iii), code review induced 162/361 (44.9%) refactoring edits. For comparison purposes, Pantiuchina et al. found that reviewing discussion triggered about 35% of refactoring edits when analyzing 551 PRs from 150 distinct projects in GitHub (Pantiuchina et al. 2020), when considering refactoring edits occurring in any PR commits (including the initial ones), regardless of what led to the refactorings (to the best of our knowledge). Therefore, although considering a distinct research design, our result corroborates with Pantiuchina et al. because it emphasizes refactoring as a relevant aiding from code reviewing in the pull-based development model (Zhu et al. 2016).

Furthermore, Table 8 summarizes 31 distinct refactoring types in light of both refactoring-inducement approaches (led by authors and induced by code review) in the validated refactorings. Accordingly, we realized that both PR authors and reviewers address issues on code quality, mainly focusing on readability, maintainability, and overall software design.

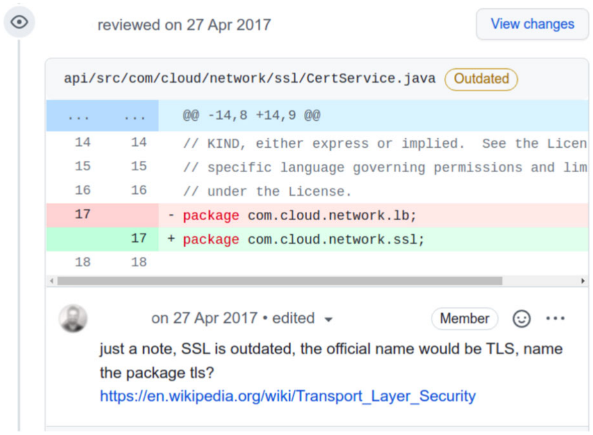

pinot-core/src/test/java/com/linkedin/pinot/core/realtime/kafka/SimpleConsumerWrapperTest.java

```
+ public class MockKafkaSimpleConsumerFactory implements KafkaSimpleConsumerFactory
+   public MockKafkaSimpleConsumerFactory(String[] hosts, int[] ports, long[] parti
+       long[] partitionEndOffsets, int[] partitionLeaderIndices, String topicName)
+     Preconditions.checkArgument(hosts.length == ports.length);
```

on 1 Sep 2016 Contributor

you could even use Assert instead of Preconditions.check() ? anyone using these mock objects should conform to what it implements ...

Fig. 4 A single review comment that induced no refactoring edit (Incubator-Pinot #479)



reviewed on 27 Apr 2017

api/src/com/cloud/network/ssl/CertService.java

```
@@ -14,8 +14,9 @@
   // KIND, either express or implied.  See the Licen
   // specific language governing permissions and lim
   // under the License.
 - package com.cloud.network.lb;
 + package com.cloud.network.ssl;
```

on 27 Apr 2017 • edited

just a note, SSL is outdated, the official name would be TLS, name the package tls?
https://en.wikipedia.org/wiki/Transport_Layer_Security

Fig. 5 A review comment that induced a *Rename Package* edit (Cloudstack #2071)

As a result, this suggests that different refactoring types occur regardless of refactoring-inducement at the PR level. In addition, for instance, the top refactoring types (*Rename Method*, *Change Type*, etc.) are potentially API-breaking changes. If these are public methods; then all clients would break after the next library release. Unless these are private methods or methods just used internally in the project. We believe reviewers would be more reluctant to recommend API-breaking changes unless these are new APIs. It would be interesting to examine in depth these top refactoring changes under these aspects: (i) Are these newly added methods or existing methods? (Maybe the methods were added in another commit in the PR) and (ii) If they are existing methods, are they public, private, or protected? Therefore, we claim that a further investigation of these edits, as future research, deserves attention since they may provide insightful and practical characterizations of refactoring-inducement, thus complementing this study.

Table 8 Refactoring types in 60 refactoring-inducing PRs (validated refactorings)

Refactoring type	Refactoring-inducement		Total
	by authors	by code review	
Rename Method	29	18	47/361 (13.019%)
Change Attribute Type	26	10	36/361 (9.972%)
Change Parameter Type	11	19	30/361 (8.311%)
Change Variable Type	10	17	27/361 (7.479%)
Change Return Type	3	19	22/361 (6.094%)
Move Class	12	7	19/361 (5.263%)
Extract Method	8	10	18/361 (4.986%)
Rename Variable	8	8	16/361 (4.432%)
Pull Up Method	5	10	15/361 (4.155%)
Extract Variable	12	2	14/361 (3.878%)
Extract And Move Method	11	3	14/361 (3.878%)
Rename Attribute	10	4	14/361 (3.878%)
Move Attribute	11	0	11/361 (3.047%)
Rename Parameter	3	8	11/361 (3.047%)

Table 8 continued

Refactoring type	Refactoring-inducement		Total
	by authors	by code review	
Push Down Attribute	8	2	10/361 (2.771%)
Rename Class	5	5	10/361 (2.771%)
Push Down Method	7	2	9/361 (2.493%)
Replace Variable W/ Attribute	2	5	7/361 (1.939%)
Move Method	5	0	5/361 (1.385%)
Extract Class	4	0	4/361 (1.108%)
Extract Superclass	2	2	4/361 (1.108%)
Pull Up Attribute	1	3	4/361 (1.108%)
Change Package	1	2	3/361 (0.831%)
Move And Rename Class	0	3	3/361 (0.831%)
Parameterize Variable	2	0	2/361 (0.544%)
Extract Interface	0	1	1/361 (0.277%)
Extract Attribute	0	1	1/361 (0.277%)
Inline Variable	1	0	1/361 (0.277%)
Merge Parameter	1	0	1/361 (0.277%)
Split Attribute	0	1	1/361 (0.277%)
Split Parameter	1	0	1/361 (0.277%)

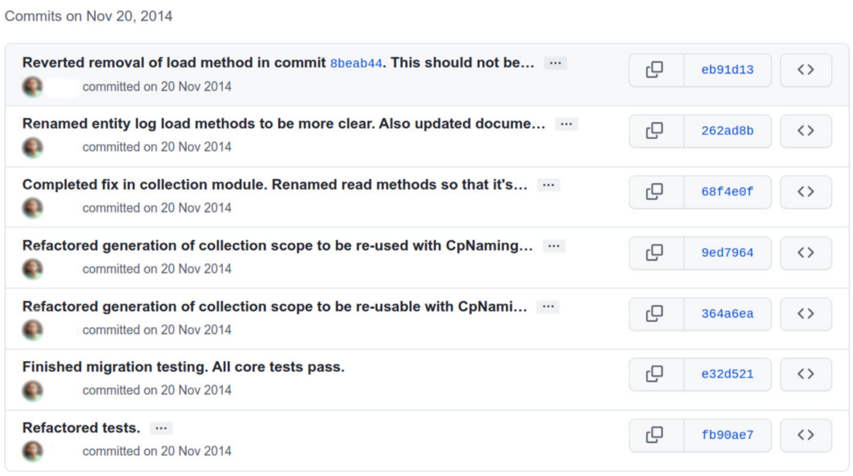
4.1.2 How Authors Document Refactoring Edits

We observed the presence of *self-affirmed refactorings* or *self-admitted refactorings* (when a refactoring is cited explicitly in a commit message by using keywords like "*Refactor...*", "*Mov...*", "*Renam...*") (AlOmar et al. 2019), as exemplified in Fig. 6, in subsequent commits of 9/60 (15.0%) refactoring-inducing PRs (Table 9). In five of them (Beam #6261, Dubbo #4099, Flink #7971, Tomee #407, Usergrid #102), all refactoring edits were led by the author. In the other ones (Avro #525, Struts #43, Tinkerpop #1110, Tomee #407), authors submitted self-affirmed refactorings to meet code review suggestions.

This result corroborates the findings of AlOmar et al., who analyzed how developers document refactorings during software evolution by exploring PR commits (including initial and subsequent ones) (AlOmar et al. 2019, 2021), given we identified a reduced number of refactoring edits documented by PR authors. They uncovered that authors use a variety of expressions to target the commits and to specify improvements (e.g., quality attributes and code smell) in the commit messages as well as we observed. Also, we found direct mentions such as "*Refactor*" (Avro #525).

4.1.3 Types of Change

Refactoring-inducing and non-refactoring-inducing PRs comprise the three *primary types of change* (adaptive, corrective, and perfective), as presented in Table 10 and detailed in our reproduction kit (Coelho 2024). To clarify, the researchers manually explored the objective of each PR (considering initial and subsequent commits, PR description, and commit mes-


Commits on Nov 20, 2014

Reverted removal of load method in commit 8beab44. This should not be... committed on 20 Nov 2014 eb91d13

Renamed entity log load methods to be more clear. Also updated docume... committed on 20 Nov 2014 262ad8b

Completed fix in collection module. Renamed read methods so that it's... committed on 20 Nov 2014 68f4e0f

Refactored generation of collection scope to be re-used with CpNaming... committed on 20 Nov 2014 9ed7964

Refactored generation of collection scope to be re-usable with CpNami... committed on 20 Nov 2014 364a6ea

Finished migration testing. All core tests pass. committed on 20 Nov 2014 e32d521

Refactored tests. committed on 20 Nov 2014 fb90ae7

Fig. 6 Example of a self-affirmed refactoring in five subsequent commits(Usergrid #102)

sages) by searching for keywords that could denote their type of changes. Thus, we classified them according to maintenance activities, as defined in guidelines (Mockus and Votta 2000; Swanson 1976). Adaptive, corrective, and perfective changes comprise adding new features (e.g., "*add*", "*new*", "*update*"), fixing faults (e.g., "*fix*", "*correct*"), and restructurings to accommodate future changes (e.g., "*simplify*", "*optimize*"), respectively. We emphasize that such a judgment was subjective and endorsed by the researchers aiming to discover potential emergent patterns from distinct types of changes toward a more comprehensive characterization of refactoring-inducing PRs.

In a few refactoring-inducing PRs, the researchers found both adaptive and corrective changes (Accumulo #151, Dubbo #2279), and corrective and perfective changes (Cloudstack #2714, Dubbo #3654). Only one non-refactoring-inducing PR comprises perfective changes related to enhancements of code documentation (Kafka #6438). Even in Java repositories, PRs can present non-Java files (e.g., Scala code), sometimes resulting in no review comment about Java files. We found such a characteristic in 15/58 (25.9%) of the non-refactoring-inducing PRs (Beam #4261 #4419 #5772 #5785 #7696 #8140, Cloudstack #2706, Flink #4055 #9451, Incubator-Iotdb #342, Kafka #5368 #6298 #6758, Parquet-Format #98, Tinkerpop #690).

Table 9 Refactoring-inducing PRs containing self-affirmed refactorings in their subsequent commits

Sample	No. of PRs	PRs
1	1/10 (10.0%)	Flink #7971
2	none	
3	2/13 (15.4%)	Beam #6261, Usergrid #102
4	6/27 (22.2%)	Avro #525, Dubbo #4099, Struts #43, Tinkerpop #1110, Tomee #89 #407
Total	*9/60 (15.0%)*	

Table 10 Type of changes by category of PRs

Sample	Type of changes		
	Adaptive	Corrective	Perfective
Refactoring-inducing PRs			
1	3/10 (20.0%)	4/10 (40.0%)	3/10 (30.0%)
2	4/10 (40.0%)	5/10 (50.0%)	1/10 (10.0%)
3	7/13 (53.8%)	3/13 (23.1%)	2/13 (15.4%)
4	11/27 (40.7%)	8/27 (29.6%)	7/27 (25.9%)
Total	24/60(40.0%)	20/60(33.3%)	13/60(21.7%)
non-Refactoring-inducing PRs			
1	1/7 (14.3%)	6/7 (85.7%)	none
2	none	7/7 (100.0%)	none
3	4/11 (36.4%)	5/11 (45.5%)	1/11 (9.1%)
4	7/18 (38.9%)	11/18 (61.1%)	none
Total	12/43(27.9%)	29/43(67.4%)	1/43(2.3%)

Hence, we considered 43/58 non-refactoring-inducing PRs when computing the number of PRs by type of change.

Our results (Table 10) confirm empirical evidence on refactoring in the presence of distinct types of change (Palomba et al. 2017). As well as Vassallo et al. (2019), we detected a higher proportion of adaptive (40.0%) and corrective (33.3%) changes. Then, we can consider adaptive, corrective, and perfective changes as opportunities for refactoring at the PR level. Note that the setting is considerably different in non-refactoring-inducing PRs, in which more than 67% of changes are corrective.

4.1.4 Self-affirmed Minor Review Comments

Aiming to investigate whether refactoring edits induced by code review commonly originate from direct/simple review comments, we explored self-affirmed minor review comments. We define it, as follows.

Definition 2 A self-affirmed minor review comment is one in which a reviewer declares it as minor.

We conducted this exploration based on empirical evidence regarding the "somewhat usefulness" of minor comments, such as those labeled as textitnit-picking or similar (Bosu et al. 2015). Accordingly, we sought potential patterns that could emerge from these self-affirmations. For instance, would they only be present in PRs that do not induce refactoring? Given that, we manually searched for the keywords "*minor*" and "*nit*" (e.g., "*LGTM, just a minor comment. Should 32 be chunkSize?*", Kafka #4574 and "*nit: Null value is encoded... –> A null value is encoded...*", Kafka #4735). Reviewers may prefix review comments with "*nit:*", so connoting that fixing a point is not mandatory but welcome.

We identified self-affirmed minor review comments in refactoring-inducing and non-refactoring-inducing PRs. We observed self-affirmed review comments that induced edits of

Rename (Brooklyn-Server #964, Flink #7945 #8620, Kafka #5784 #6848), *Split* (Brooklyn-Server #1049), and *Extract* (Flink #8620, Kafka #4735). Therefore, we distinguished no association between self-affirmed minor review comments and refactoring-inducement.

4.1.5 Code Review Bots

The researchers identified that 7/60 (11.7%) of refactoring-inducing PRs (Flink #7945 #7970 #7971 #8222 #8620 #9143, Struts #43) and 2/58 (3.4%) of non-refactoring-inducing PRs (Flink #9451, Hadoop #942) ran a *code review bot*. It is easily detectable since it leaves comments in a PR, including the bot commands. For instance, the Apache *flinkbot*[2] checks the PR description, whether a PR needs attention from a specific reviewer, the architecture, and the overall code quality. Thus, we observed no association between running a code review bot and refactoring inducement.

Finding 1: We found 50% of refactoring-inducing PRs, in which code review solely induced refactoring edits, in 118 Apache's merged PRs. The refactorings happen in PRs consisting of distinct types of change (adaptive, corrective, and perfective). We observed no association between self-affirmed minor review comments and refactoring inducement. Self-affirmed refactorings and code review bots are not frequently applied.

4.1.6 Experience of PR Authors and Reviewers

We explored the experience of PR authors and reviewers, by examining their number of contributions, in GitHub profiles (e.g., a code review, the creation of a pull request, and the submission of a commit), computed as we argued in Section 3.5. Table 11 presents the average and median statistics computed for the experience of authors and reviewers in the purposive samples.

It is noteworthy that, when analyzing the distribution of authors' and reviewers' experience in both groups of PRs, we visually realized that authors of refactoring-inducing PRs are less experienced than authors of non-refactoring-inducing PRs, whereas reviewers of refactoring-inducing PRs are slightly more experienced than reviewers of non-refactoring-inducing PRs. Then, for testing these impressions, we formulated the hypotheses described in Section 3.5.

Table 12 presents the computed statistics and p-value for each applied statistical test when analyzing authors' experience, as the hypotheses $H1_0$ (there is no difference between authors' experience in refactoring-inducing and non-refactoring-inducing PRs) and $H1_a$ (authors of refactoring-inducing PRs are less experienced than authors of non-refactoring-inducing PRs). The computed 95% CI [-817.0, 514.2], by bootstrapping resample, of the difference in medians contains 0, thus we found no statistically significant difference in authors' experience in refactoring-inducing and non-refactoring-inducing PRs. Accordingly, based on the Mann-Whitney one-sided test (U = 1,215.50, $p < .05$), *there is no statistical evidence that authors of refactoring-inducing PRs are less experienced than authors of non-refactoring-inducing PRs*.

Table 13 shows the computed statistics and p-value for each applied statistical test when analyzing reviewers' experience, in line with hypotheses $H2_0$ (there is no difference between reviewers' experience in refactoring-inducing and non-refactoring-inducing PRs) and $H2_a$ (reviewers of refactoring-inducing PRs are more experienced than reviewers

[2] https://github.com/flinkbot

Table 11 Statistics of authors' and reviewers' experience

Sample	Statistics	
	Authors	Reviewers
Refactoring-inducing PRs		
1	886.1 (875.2), 550 (1,118)	3,162.3 (3,674.2), 2,205.5 (3,747)
2	3,854.1 (8,884.3), 1,082 (1,064)	2,926.8 (4,013.6), 1,278 (1,037)
3	775.6 (1,586.2), 115 (663)	3,563.7 (8,059.1), 853 (2,551)
4	2,874.4 (4,753.7), 946 (2,851)	2,797.3 (4,753.3), 1,397 (2,727)
Total	2,224.9 (4, 867.4), 510 (1,289)	3,046.4 (5,671.4), 1,275 (2,928.5)
non-Refactoring-inducing PRs		
1	4,025.1 (9,571), 391 (966)	3,348.9 (4,138.2), 1,460.5 (4,089)
2	2,637.8 (4,193.7), 1,285 (2,130)	1,890.1 (1,640.5), 1,639 (3,371)
3	1,612.3 (2,783.6), 158 (1,104)	3,653.8 (9,027.5), 777 (827)
4	3,828.1 (8,378.1), 808 (2,744)	3,141.4 (6,622.8), 943 (2,510)
Total	3,146.6 (6,826.1), 1,035 (1,979)	3,146.6 (6,826.1), 1,035 (1,979)

where <number> (<number>), <number> (<number>) denotes the average (standard deviation) and median (interquartile range), respectively

of non-refactoring-inducing PRs). The computed 95% CI [-78.9, 803.6], by bootstrapping resample, of the difference in medians contains 0, denoting no statistically significant difference in reviewers' experience in refactoring-inducing and non-refactoring-inducing PRs. Based on the Mann-Whitney one-sided test (U = 4,521.0, $p < .05$), *there is no statistical evidence that reviewers of refactoring-inducing PRs are more experienced than reviewers of non-refactoring-inducing PRs.*

Table 14 displays the computed statistics and p-value for each applied statistical test for the hypotheses $H3_0$ (there is no difference between the experience of authors and reviewers in refactoring-inducing PRs) and $H3_a$ (authors are less experienced than reviewers in refactoring-inducing PRs). Based on the Mann-Whitney one-sided test (U = 2,616.0, $p < .05$), *there is statistical evidence that authors are less experienced than reviewers in refactoring-inducing PRs.*

Table 15 presents the computed statistics and p-value for each applied statistical test for hypotheses $H4_0$: (there is no difference between the experience of authors and reviewers in non-refactoring-inducing PRs) and $H4_a$ (authors are more experienced than reviewers in non-refactoring-inducing PRs). Based on the Mann-Whitney one-sided test (U = 1,395.5, $p < .05$), *there is no statistical evidence that authors are more experienced than reviewers in non-refactoring-inducing PRs.*

Table 12 Statistical tests output - authors' experience in refactoring-inducing and non-refactoring-inducing PRs

Statistical test	Statistic	p-value
Shapiro-Wilk	Refactoring-inducing PRs: 0.48	$2.64 \times e^{-13}$
	Non-refactoring-inducing PRs: 0.49	$5.92 \times e^{-11}$
Levene's	0.61	0.44
Mann-Whitney U	1,215.5	0.26

Table 13 Statistical tests output - reviewers' experience in refactoring-inducing and non-refactoring-inducing PRs

Statistical test	Statistic	p-value
Shapiro-Wilk	Refactoring-inducing PRs: 0.52	$8.15 \times e^{-18}$
	Non-refactoring-inducing PRs: 0.44	$1.16 \times e^{-15}$
Levene's	0.0089	0.92
Mann-Whitney U	4,521.0	0.44

In summary, we found a significant difference between authors' and reviewers' experience within refactoring-inducing PRs, suggesting that authors are less experienced than reviewers in such a context – a pattern identified in 42/60 (70%) of refactoring-inducing PRs. Since we found no statistical evidence that the number of reviewers is related to refactoring-inducement (Coelho et al. 2021), this new finding denotes a relevant motivating factor behind refactoring-inducing PRs. Based on this finding, we conjecture that less problem-prone code tends to give origin to non-refactoring-inducing PRs more often, which can be partially explained by the authors' experience. We contextualize the importance of reviewers to characterize refactoring-inducing PRs in Section 4.3. Even so, we claim that a further investigation of the content of contributions could provide a better understanding of the relationship between the experience of authors/reviewers and refactoring-inducing PRs.

Finding 2: The experience of the PR author, inferred from the number of contributions, is a motivating factor behind refactoring-inducing PRs.

In addition, we identified a few particular scenarios when considering experience concerning the number of authors' and reviewers' contributions. First, code review induced refactoring edits in Fluo #837, although the number of the author's contributions (3,127) is greater than the individual number of contributions of its reviewers (3,040 and 2,192). The same occurs in Dubbo #3174. We conjecture that the aggregate of experiences might explain those cases. Code review also induced refactorings in PRs in which the authors have a higher number of contributions than their reviewers, such as occurs in Brooklyn-Server #964 and Tinkerpop #1110 (13,955 and 14,288, for authors; 1,660/2,933 and 24/556, for reviewers). Accordingly, this seems counter-intuitive in contrast to previous findings with two reviewers providing a more effective code review (Rigby et al. 2012). Second, we observed that a few PR authors led refactoring edits, even when their number of contributions was shorter in contrast to reviewers (e.g., Kafka #4574). We believe that an in-depth examination of the code might help to explain that scenario.

Table 14 Statistical tests output - authors' and reviewers' experience in refactoring-inducing PRs

Statistical test	Statistic	p-value
Shapiro-Wilk	Refactoring-inducing PRs: 0.48	$2.64 \times e^{-13}$
	Non-refactoring-inducing PRs: 0.52	$8.15 \times e^{-18}$
Levene's	0.42	0.52
Mann-Whitney U	2,616.0	0.002

Table 15 Statistical tests output - authors' and reviewers' experience in non-refactoring-inducing PRs

Statistical test	Statistic	p-value
Shapiro-Wilk	Refactoring-inducing PRs: 0.49	$5.92 \times e^{-11}$
	Non-refactoring-inducing PRs: 0.44	$1.16 \times e^{-15}$
Levene's	0.04	0.84
Mann-Whitney U	1,395.5	0.92

4.2 How are Review Comments Characterized in Refactoring-inducing and Non-refactoring-inducing PRs?

We structure this answer considering code review in non-refactoring-inducing PRs versus three dimensions of refactoring-inducing PRs: those with refactorings led exclusively by authors (16/60, 26.7%), those with refactorings induced solely by code review (30/60, 50.0%), and those with refactorings both led by the authors and induced by code review (14/60, 23.3%). Particularly, in refactoring-inducing PRs Beam #6261, Dubbo #3654, Flink #8620, and Kafka #4757, we found the presence of floss refactoring to accommodate new tests in the three first ones and a new feature in a class in the last one – all suggested by reviewers. Thereby, we judged the associated refactoring edits as led by the PR authors.

4.2.1 Review Comments in Refactoring-inducing PRs with Refactorings Led Exclusively by Authors

Table 16 summarizes the emerged characteristics of code reviews in refactoring-inducing PRs, with refactoring edits submitted entirely by the authors. The authors properly provide explanations to reviewers' questions not related to potential refactorings. A usual review comment in this case:

– address *code aesthetics* (e.g., "*whitespace between braces and other symbol.*" in Flink #7971),
– present questions on *simple issues regarding code logic* in contrast to those in PRs with refactorings induced by code review (e.g., "*Should 32 be chunkSize?*" in Kafka #4574), and
– sometimes, give *reasons* and *suggestions on code logic*, using expressions such as "*could we use...?*", "*use...*", "*can be replaced with...*" (e.g., "*token.sum() can be replaced with getToken()*" in Dubbo #3654).

In summary, we identified review comments addressing code aesthetics in 2/16 PRs (Dubbo #2445, Flink #7971), questioning simple issues on code logic in 7/16 PRs (Accumulo #151, Beam #4460 #6261, Kafka #4574 #6657, Logging-log4j2 #213, Samza #1030), and suggesting improvements to the code in 10/16 PRs (Beam #6261, Dubbo #2445 #3654 #4099, Incubator-Pinot #479, Kafka #5423 #6657, Tinkerpop #893, Tomee #89, Usergrid #102).

4.2.2 Review Comments in Refactoring-inducing PRs with Refactorings Induced Exclusively by Code Review

Table 17 indicates the emerged characteristics of reviews in refactoring-inducing PRs with refactoring edits induced only by code review. We observed that review comments:

Table 16 Characteristics of review comments in refactoring-inducing PRs (Refactorings Led by the Authors, i.e., refactorings suggested only by commit authors)

Characteristic	Examples (PRs)
Addressing code aesthetics	Code format (“*format your code, pls.*”, Flink #7971)
Questioning simple issues on code logic	Access by name or index (“*Don’t we still use this for Dataflow in the NonFnApi mode (which passes connections by index)?*”, Beam #4460)
	Error in a conditional statement (“*if the endTime is less than 0 wouldn’t we want to throw an exception?*”, Kafka #6657)
	Treatment of specific values (“*Does it make sense to fail if the partition is empty?*”, Samza #1030)
	Dealing with potential failure (“*Logback is EPL/LGPLv2.1. Not sure if the licensing is compatible here.*”, Logging-Log4j2 #213)
	Method calls (“*Is the mockstatic call why we’re adding powermock here?”*, questioning the effect of a call, Beam #6261)
Suggesting improvements to the code	Adding case tests (“*Add a unit test for this case?*”, Kafka #5423) Method calls (“*nit: use Objects.requireNonNull with the same message here and below*” proposing a method replacement, Kafka #6657)
	Adding code documentation (“*Could you add detail on the documentation? perhaps some small examples, etc?*”, Beam #6261)
	Change the content of test files (“*Could we put this into a resource file instead of having it hang around in the test?*”, Usergrid #102)
	Use of assertion (“*you could even use Assert instead of Preconditions.check()?”*, Incubator-Pinot #479)

– ask questions about *code logic* (e.g., “*should we simply keep Timeout instance (instead of keeping it in a map)?*”, Dubbo #3299),
– suggest *improvements* to the code (e.g., “*Maybe put the table name in a variable, so the string parameter is more obviously the table name in this API call, rather than something else.*”, Accumulo-Examples #19),
– and provide warnings on *good development practices* (e.g., “*Constants, private static final, are usually all caps: MAX_RETRY_COUNT. Within the context of a retryAnalyzer you could get away with count and MAX. They are both private.*”, Brooklyn-Server #1049).

In short, we realized review comments questioning issues on code logic in 7/30 PRs (Beam #4407, Dubbo #3174 #3299, Fluo #837, Kafka #4757 #5784, #Tika 234), suggesting improvements to the code in 21/30 PRs (Beam #4458, Brooklyn-Server #964, Cloudstack #2071 #3454, Dubbo #3257, Flink #8620 #9143, Kafka #4796 #5194 #5590 #6853 #7132, Knox #74, Samza #1051, Servicecomb-Java-Chassis #346, Sling-Org-Apache-Sling-Feature-Analyser #16, Struts #43, Tika #234, Tinkerpop #1110, Tomee #275 #407),

Table 17 Characteristics of review comments in refactoring-inducing PRs (Refactorings Induced by Code Review, i.e., refactorings suggested only by code reviewers)

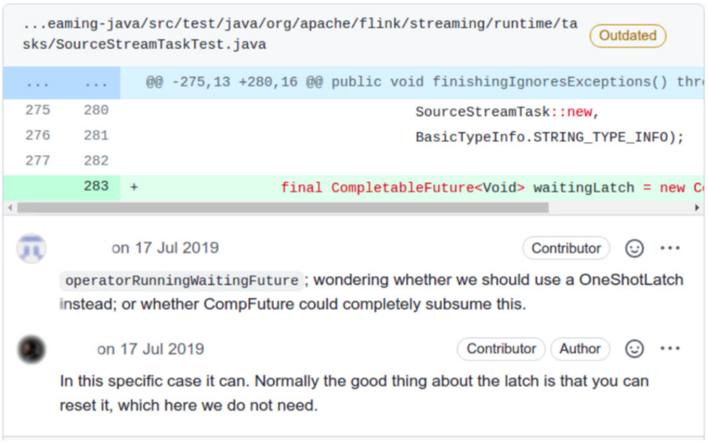
Characteristic	Examples (PRs)
Questioning issues on code logic	Use of specific types ("*Be careful that this attachment TIMEOUT_ FILTER_START_TIME will be passed throughout the RPC chain because of the drawback of RpcContext.*", Dubbo #3174)
	Method calls ("*Is it safe to call toString here on arbitrary bytes?*", Fluo #837)
Suggesting improvements to the code	Refactoring ("*Can we pass in a Path instead of a String for keystorePath?*", Knox #69)
	Adding code documentation ("*it's better to write new comments in english.*" asking for comments in English, Servicecomb-Java- Chassis #346)
	Adding case tests ("*nit: add also another case for something not ending in ConfigProvider?*", Kafka #5194)
	Adding an exception handling ("*Same as with deserialize. Should we consider throwing an exception?*", Kafka #5590)
	Discard a method, by providing explanations about code design ("*I don't think there is a need for this to be left. The truncate length should always be included when getting metrics.*", Incubator-Iceberg #254)
	Issues on GUI layout ("*it's better now, but can you make them centered left, they seem to be bottom left now.*", Cloudstack #3454)
Warnings on good development practices	Code conventions ("*Constants, private static final, are usually all caps*", Brooklyn-Server #1049)
	Switch-case against multiple if-else ("*it would be better to use a switch case instead of multiple if else, especially for comparing multiple enums*", Cloudstack #2833)

and providing warnings on good development practices in 4/30 PRs (Brooklyn-Server #1049, Cloudstack #2833, Fluo #837, Knox #69).

Uncertain review comments (like "*I'm not sure ...*", "*wondering if...*", "*just thinking loud here ...*", and "*As far as I remember ...*") are not usually welcomed; for instance, a reviewer wondering if another strategy is appropriate to deal with code logic led to discussion but no effect (Flink #9143, Kafka #7132), as illustrated in Fig. 7. Moreover, using *embedded code* in review comments is only appreciated when it is a suggestion instead of an imposition. As an example, Dubbo #3299's author ignored a review comment containing an embedded code (for treating a variable status). We speculate that such a scenario may affect code ownership, so explaining the behavior of PR authors (more examples in Section 4.6).

4.2.3 Review Comments in Refactoring-inducing PRs with Refactorings both Led by Authors and Induced by Code Review

In refactoring-inducing PRs, comprising refactoring edits both led by the authors and induced by code review, the review comments present characteristics that already emerged from exploring the refactoring-inducement in PRs (Table 18). Uncertain review comments remain in such a scenario, where the sentences include terms such as "*Looks like...*" and "*not sure if...*" in Incubator-Iceberg #183. Based on Tables 16, 17 and 18, when code review induces



```
...eaming-java/src/test/java/org/apache/flink/streaming/runtime/tasks/SourceStreamTaskTest.java
```

Outdated

```
@@ -275,13 +280,16 @@ public void finishingIgnoresExceptions() thr
                                SourceStreamTask::new,
                                BasicTypeInfo.STRING_TYPE_INFO);

+                final CompletableFuture<Void> waitingLatch = new C
```

on 17 Jul 2019 Contributor

`operatorRunningWaitingFuture` ; wondering whether we should use a OneShotLatch instead; or whether CompFuture could completely subsume this.

on 17 Jul 2019 Contributor Author

In this specific case it can. Normally the good thing about the latch is that you can reset it, which here we do not need.

Fig. 7 An uncertain review comment example (Flink #9143)

Table 18 Characteristics of review comments in refactoring-inducing PRs (Refactorings both Led by the Authors and Induced by Code Review, i.e., refactorings suggested both by commit authors and code reviewers)

Characteristic	Examples (PRs)
Addressing code aesthetics	Indentation and code format ("*Nit: extra blank line.*", Incubator-Iceberg #119)
Questioning issues on code logic	Use of specific types ("*Collection is useless.*", Dubbo #2279) Error in a conditional statement ("*nit: do we want to consider setting producer to null here as well if eosEnabled?*", Kafka #5501)
	Treatment of specific values ("*Do we need use the next begin offset in other PullStatus?*", Rocketmq-Externals #45)
	Method calls ("*Can you not use .withClientSsl-Support(). withClientSaslSupport()?*" suggests an optimization in a method call, Kafka #4757)
Suggesting improvements to the code	Refactoring ("*After adding the time conversions to this method the name (or the behavior) is really misleading.*", Avro #525)
	Adding an exception handling ("*I think we should not easily catch on Throwable for simplicity but instead it is clear here that we should only expect IOException or ClassCastException.*", Flink #7970)
	Adding code documentation ("*Gets -> Get, let's to be consistent with javadoc of other methods.*" to make it consistent with other classes, Flink #8222)
	Adding case tests ("*renameTable too? maybe add test?*", Flink #8222)
Warnings on good development practices	Use of methods in tests ("*Tests shouldn't use methods in other tests because they are hard to keep track of.*", Incubator-Iceberg #183)
	Use of global instances ("*global instance. it's better do not use this directly. you can save a reference in filter when UT create a new instance for it.*", Servicecomb-Java-Chassis #678)
	Instructions to create integration tests ("*Adding the exception is fine, but you can just throw it directly: throw new OffsetOutOfRangeException(...) Not need to assign it to variable first :)*", Kafka #5946)

refactoring edits, we realize that review comments deal with more complex issues regarding code logic and concerns on good development practices.

In summary, we found review comments addressing code aesthetics only in Incubator-Iceberg #119, questioning issues on code logic in 5/14 PRs (Dubbo #2279, Flink #8222, Kafka #4735 #5501, Rocketmq-Externals #45), suggesting improvements to the code in 6/14 (Avro #525, Flink #7165 #7970 #8222, Kafka #4735 #6848), and warning on good development practices in 5/14 PRs (Flink #7945, Incubator-Iceberg #119 #183, Kafka #5946, Servicecomb-Java-Chassis #678).

Finding 3: In refactoring-inducing PRs in which code review induces refactoring edits, review comments address major issues on code logic, major improvements to the code (including refactorings directly), and warnings on good development practices.

Finding 4: In refactoring-inducing PRs in which the authors lead refactoring edits, the review comments address code aesthetics, minor issues in code logic, and minor improvements to the code.

4.2.4 Review Comments in Non-refactoring-inducing PRs

Review comments in non-refactoring-inducing PRs directly address minor issues on code logic through questions and suggestions (Table 19). We also found review comments including embedded code as an imposition for substituting the code ("*should we change...*"), with no effect (Dubbo #4870 and Kafka #5111). However, such a structure of review comment

Table 19 Characteristics of review comments in non-refactoring-inducing PRs

Characteristic	Examples (PRs)
Examining code aesthetics	Code format ("*Code formatting for this section of code is different (spaces used)*", Cloudstack #3430)
Addressing issues on code logic	Error in conditional statements ("*I think b==0?true:false can be replaced with b==0.*", Fluo #929)
	Exception handling ("*IllegalStateException is better?*", Servicecomb-Java-Chassis #691)
	Method calls ("*What about overriding the method getCause to return this value here?*" suggesting a method overriding, Cloudstack #2714)
	Reverting of changes ("*L116 and 118 also have changes that should be reverted.*", Kafka #5111)
	Removing of code fragments ("*why those code are removed? are we still need -Djava.net.preferIPv4Stack=true after this change are made?*", Dubbo #3317)
	Requiring additional states for sessions ("*I wonder if we could include any additional state from the session itself to get closer to the root of the problem.*", Kafka #6427)
Suggesting improvements to the code	Adding assertion ("*nit: Add assertion for a part of the exception message?*", Beam #6317)
	Adding case tests ("*can you implement a marvin test for the test-case.*", Cloudstack #3276)

Table 19 continued

Characteristic	Examples (PRs)
	Adding code documentation ("*It might nice to add some comments describing the difference between informational and comparison.*", Accumulo-Testing #21)
	Fixing code documentation ("*This comment could be improved to explain why the client needs to be left open here. I don't think it's merely a matter of 'simplicity'.*", Accumulo-Examples #50)
	Alternatives to code fragments ("*we should not copy the code from the old addGlobalStore() but rather call the old addGlobalStore() passing the generated names.*", Kafka #4430)
	Alternatives to error/output messages ("*Suggest using 'LOGGER. error("Cannot get PID of IoTDB process because ", e);*", Incubator-Iotdb #67
	Fixing a typo ("*typo: terminated*", Flink #91)
	Use of better argument values ("*I think this may cause an issue, because the vlan.getVlanTag() could be something like a range 100-200, so if vlanId is 101 the check should be done in the range.*", Cloudstack #3430)

is welcomed when the author asks for support from the reviewer, as it occurs in Fluo #929. Therefore, using embedded code is appreciated in the form of a suggestion.

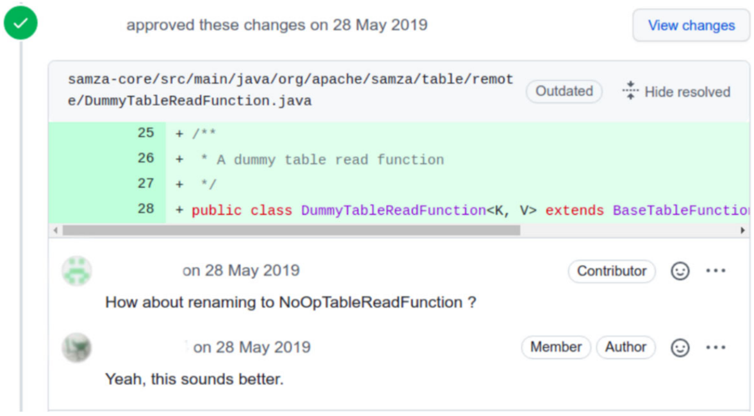
Those minor issues denote subjects having no relation to refactoring, such as fixing an error in conditional statements and using the correct range of variables. Usually, the authors provide answers to the reviewers, even in the presence of discussions, resulting in no effect (Beam #7696, Brooklyn-server #411, Cloudstack #2553 #2714 #3276, Dubbo #3184 #3317 #3748 #4870, Incubator-Iotdb #67, Kafka #5111 #6818, Servicecomb-Java-Chassis #691 #744, Tinkerpop #282 #524).

Finding 5: In non-refactoring-inducing PRs, review comments address code aesthetics, marginal issues on code logic, and marginal improvements to the code.

4.3 What are the Differences Between Refactoring-inducing and Non-refactoring-inducing PRs, in Terms of Review Comments?

We realized that review comments in refactoring-inducing and non-refactoring-inducing PRs are different concerning the following criteria:

Addressed Issues Based on Tables 16, 17, 18 and 19, review comments addresses *code aesthetics*, *code logic*, and *improvements* in both refactoring-inducing and non-refactoring-inducing PRs. From Findings 3–5, we recognize that review comments concern trivial issues on code logic and suggestions of minor improvements to the code occurred more in non-refactoring-inducing PRs than in refactoring-inducing PRs. In a typical PR with refactorings induced by code review, usually, review comments point out a structural problem, as in Samza #1051 (Fig. 8), in which the author applied a *Rename Class* after a review comment. Reviewers ask questions regarding issues of minor scope in non-refactoring-inducing PRs (in this case, a typical review comment asks on alternatives already implemented by the author, e.g., a question on an overloaded method in Beam #6050, Fig. 9), whereas they deal



approved these changes on 28 May 2019

View changes

samza-core/src/main/java/org/apache/samza/table/remote/DummyTableReadFunction.java

Outdated

Hide resolved

```
+ /**
+  * A dummy table read function
+  */
+ public class DummyTableReadFunction<K, V> extends BaseTableFunctio
```

on 28 May 2019 Contributor

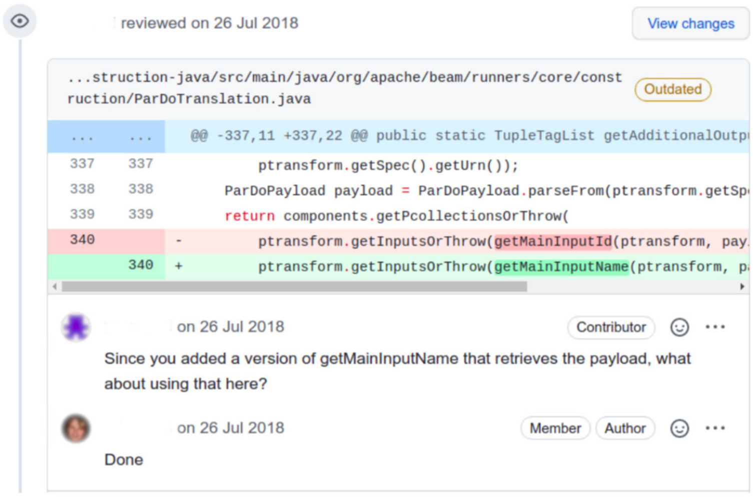
How about renaming to NoOpTableReadFunction ?

on 28 May 2019 Member Author

Yeah, this sounds better.

Fig. 8 A typical suggestion of *Rename* edit (Samza #1051)

with major issues that can induce refactoring in refactoring-inducing PRs (in this case, a typical review comment addresses more complex structural issues, as in Tinkerpop #1110 when concerning a potential *Extract Method*, Fig. 10). Reviewers provide suggestions of minor improvements to the code (e.g., alternatives to output messages) in non-refactoring-inducing PRs, whereas directly suggest refactoring (e.g., an alternative to a method signature) in refactoring-inducing PRs. The same patterns emerged when comparing review comments in refactoring-inducing PRs, where authors led the refactoring edits (Table 16), and non-refactoring-inducing PRs (Table 19). In such a context, we found more questions on minor issues on code logic in non-refactoring-inducing PRs (e.g., a doubt concerning a method call) than in refactoring-inducing PRs (e.g., a question on the effect of a method call); and suggestions of minor improvements in non-refactoring-inducing PRs (e.g., proposing the use of "*A*", "*B*", instead of "*X*", "*Y*" as method arguments) than in refactoring-inducing PRs (e.g.,



reviewed on 26 Jul 2018

View changes

...struction-java/src/main/java/org/apache/beam/runners/core/construction/ParDoTranslation.java

Outdated

```
@@ -337,11 +337,22 @@ public static TupleTagList getAdditionalOutp
        ptransform.getSpec().getUrn());
    ParDoPayload payload = ParDoPayload.parseFrom(ptransform.getSp
    return components.getPcollectionsOrThrow(
-       ptransform.getInputsOrThrow(getMainInputId(ptransform, pay
+       ptransform.getInputsOrThrow(getMainInputName(ptransform, p
```

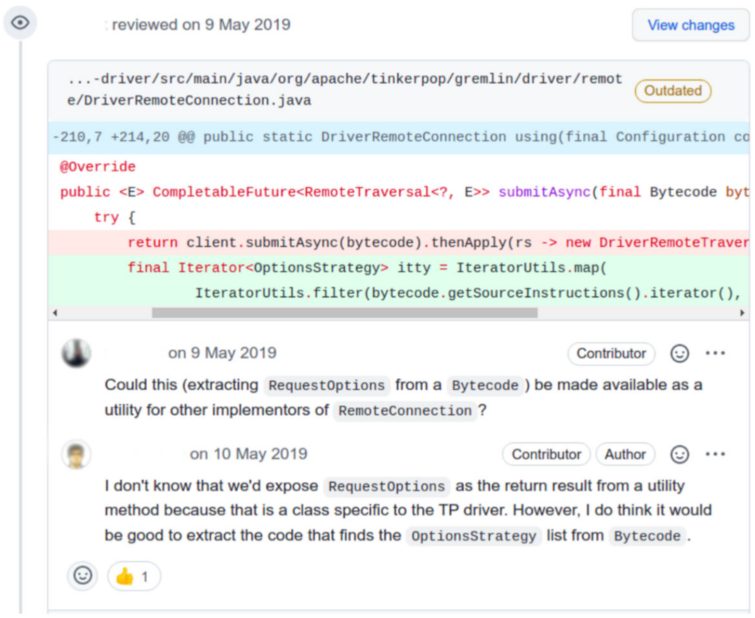
on 26 Jul 2018 Contributor

Since you added a version of getMainInputName that retrieves the payload, what about using that here?

on 26 Jul 2018 Member Author

Done

Fig. 9 A review comment on a minor scope issue (Beam #6050)


reviewed on 9 May 2019

View changes

...-driver/src/main/java/org/apache/tinkerpop/gremlin/driver/remote/DriverRemoteConnection.java

Outdated

```
-210,7 +214,20 @@ public static DriverRemoteConnection using(final Configuration co
@Override
public <E> CompletableFuture<RemoteTraversal<?, E>> submitAsync(final Bytecode byt
    try {
        return client.submitAsync(bytecode).thenApply(rs -> new DriverRemoteTraver
        final Iterator<OptionsStrategy> itty = IteratorUtils.map(
                IteratorUtils.filter(bytecode.getSourceInstructions().iterator(),
```

on 9 May 2019 Contributor

Could this (extracting `RequestOptions` from a `Bytecode`) be made available as a utility for other implementors of `RemoteConnection`?

on 10 May 2019 Contributor Author

I don't know that we'd expose `RequestOptions` as the return result from a utility method because that is a class specific to the TP driver. However, I do think it would be good to extract the code that finds the `OptionsStrategy` list from `Bytecode`.

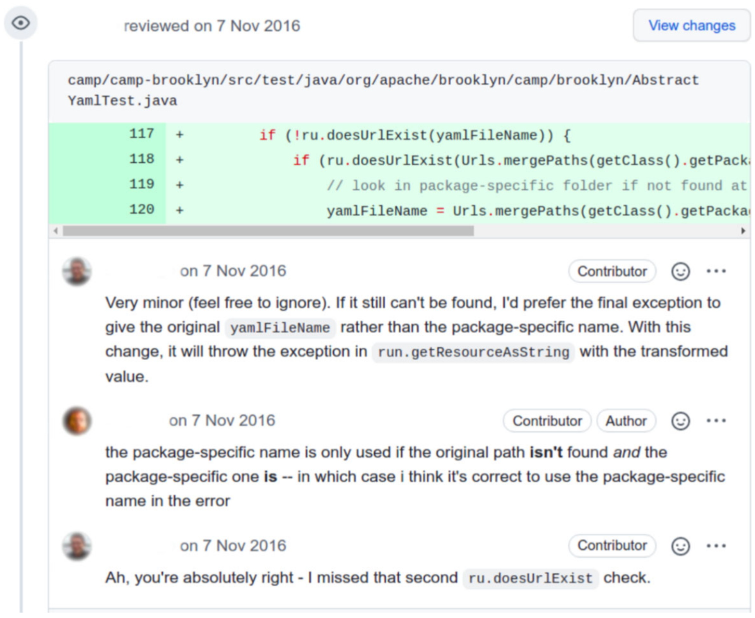
👍 1

Fig. 10 A review comment on a major scope issue (Tinkerpop #1110)

suggesting a method replacement). We found warnings on good practices of development only in refactoring-inducing PRs that, in turn, may induce refactorings (Tables 17 and 18).

Discussion We found reviewing discussion in 12/60 (20.0%) refactoring-inducing PRs and 15/43 (34.8%) non-refactoring-inducing PRs, where 43 indicates the number of non-refactoring-inducing PRs that contain reviewed Java files. We also detected reviewing discussion in two non-refactoring-inducing PRs (Dubbo #3447 #3184), in which reviewers agree with each other when addressing code issues (those issues concern no structural changes, so no inducing refactorings). In refactoring-inducing PRs, reviewing discussion tends to arise from either warning on good practices of development (authors may require explanations due to no knowledge of such practices, Kafka #4757) or questions/opinions concerning code logic, such as addressing potential failures and alternatives to a specific decision making (e.g., change a package name to meet project standards may break external plugins to use of a service in Servicecomb-Java-Chassis #678). We realize that authors properly provide clarifications for questions/opinions of reviewers. The reviewing discussion in non-refactoring-inducing PRs usually fits a pattern: everything begins with a review comment suggesting some change, followed by direct arguments from the author to refute it. As a result, the author's view prevailed (as exemplified in Fig. 11).

Structure of the Content of Review Comments We observed that, usually, review comments are more polite in refactoring-inducing PRs than in non-refactoring-inducing PRs. They consist of sentences that incorporate expressions like "*Maybe we should* ..." and "*How about ...?*", thus triggering the authors to think better concerning the code reviewed, which tends to result in refactoring. Yet, we identify a lot of review comments that impose a change in non-refactoring-inducing PRs by using expressions such as "... *should be* ...", which tends to have no effect.


reviewed on 7 Nov 2016

camp/camp-brooklyn/src/test/java/org/apache/brooklyn/camp/brooklyn/AbstractYamlTest.java

```
+         if (!ru.doesUrlExist(yamlFileName)) {
+             if (ru.doesUrlExist(Urls.mergePaths(getClass().getPack
+                 // look in package-specific folder if not found at
+                 yamlFileName = Urls.mergePaths(getClass().getPacka
```

on 7 Nov 2016

Very minor (feel free to ignore). If it still can't be found, I'd prefer the final exception to give the original `yamlFileName` rather than the package-specific name. With this change, it will throw the exception in `run.getResourceAsString` with the transformed value.

on 7 Nov 2016

the package-specific name is only used if the original path **isn't** found *and* the package-specific one **is** -- in which case i think it's correct to use the package-specific name in the error

on 7 Nov 2016

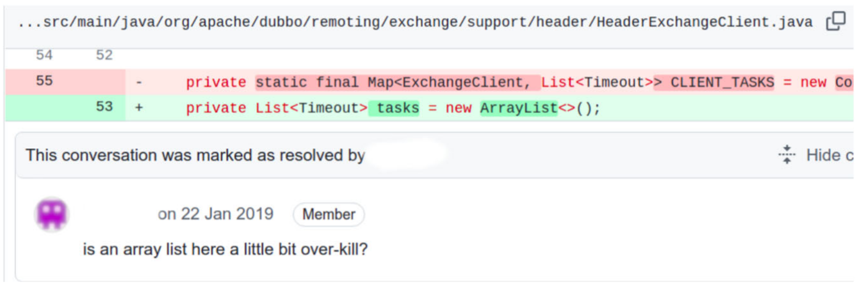
Ah, you're absolutely right - I missed that second `ru.doesUrlExist` check.

Fig. 11 A typical discussion in non-refactoring-inducing PRs (Brooklyn-Server #411)

Experience of Reviewers By counting the number of contributions of authors and reviewers, we realized that review comments are submitted by reviewers more experienced than authors in refactoring-inducing PRs (Tables 11 and 14). Accordingly, in non-refactoring-inducing PRs, the experience of authors reflects in both characteristics of their code and their ability to provide clarifications and win discussions; whereas the experience of reviewers in refactoring-inducing PRs reflects in precise review comments that may induce or inspire authors to refactor the code. Yet, we analyzed the experience of authors and reviewers in the three subgroups of refactoring-inducing PRs (with refactoring led by the author, with refactorings induced by code review, and with refactorings both led by the author and induced by code review). We noticed that, in all subgroups, the reviewers are more experienced than the authors. Thus, based on Finding 2, we conjecture that less problem-prone code seems to give origin to non-refactoring-inducing PRs more often, which can be partially explained by the experience of their authors.

Finding 6: The addressed issues and content structure of review comments (polite and precise), usually leveraged due to a higher experience (inferred from the number of contributions) of a reviewer concerning an author, are motivating factors behind refactoring-inducing PRs.

It is noteworthy the main difference between review comments in refactoring-inducing and non-refactoring-inducing PRs lies in the following evidence: non-refactoring-inducing PRs submit more well-structured and high-quality code (that is, adherence to object-oriented principles, readability, and maintainability) than refactoring-inducing PRs – what is perceived by review comments that concentrate effort in trivial issues, in the form of suggestions that do not require changing the code. On the other hand, review comments tend to induce changes in refactoring-inducing PRs politely. To exemplify, consider issues on method calls: reviewers



```
...src/main/java/org/apache/dubbo/remoting/exchange/support/header/HeaderExchangeClient.java
55  -     private static final Map<ExchangeClient, List<Timeout>> CLIENT_TASKS = new Co
53  +     private List<Timeout> tasks = new ArrayList<>();
```

This conversation was marked as resolved by

on 22 Jan 2019 Member

is an array list here a little bit over-kill?

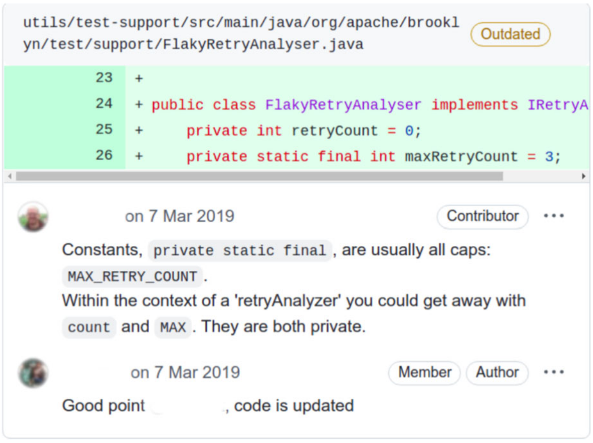
Fig. 12 A direct review comment that induced a *Change Variable Type* edit (Dubbo #3299)

typically alert on the method design in refactoring-inducing PRs (e.g., "*Tests shouldn't use methods in other tests because they are hard to keep track of.*", Incubator-Iceberg #183) whereas in non-refactoring-inducing PRs, they usually indicate minor suggestions (e.g., "*What about overriding the method getCause to return this value here?*", Cloudstack #2714).

In practice, precise and polite review comments tend to induce refactoring edits because they shed light on code issues of significant scope, therefore causing authors to embrace refactoring suggestions or get inspired by them. In this context, polite denotes review comments that make suggestions rather than impositions, while precise expresses opposition to uncertain review comments. Thus, the benefits of refactoring the code become apparent already in code review time, either through a direct suggestion of refactoring (Fig. 12), a rationale (Fig. 13), or a warning on good development practices (Fig. 14). In those examples, the review comments trigger a new author's perception, causing their agreement with the suggestions. In such a scenario, the experience of the PR reviewer(s) became crucial.

Therefore, our analysis suggests there is a strong relation between code review and refactoring edits because polite and precise review comments trigger refactorings at the PR level, thus corroborating with previous works regarding the effectiveness of code review on refactoring the code (Pantiuchina et al. 2020; Paixão et al. 2020).

Fig. 13 A review comment providing a reason (code conventions) that induced a *Split Attribute* edit (Brooklyn-Server #1049)



```
utils/test-support/src/main/java/org/apache/brooklyn/test/support/FlakyRetryAnalyser.java  Outdated
+
+ public class FlakyRetryAnalyser implements IRetryA
+     private int retryCount = 0;
+     private static final int maxRetryCount = 3;
```

on 7 Mar 2019 Contributor

Constants, `private static final`, are usually all caps: `MAX_RETRY_COUNT`.
Within the context of a 'retryAnalyzer' you could get away with `count` and `MAX`. They are both private.

on 7 Mar 2019 Member Author

Good point , code is updated

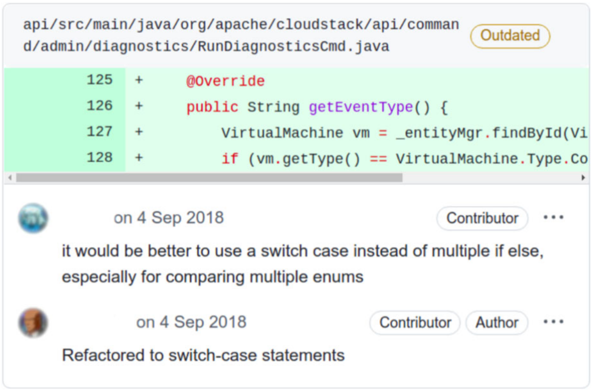


Fig. 14 A warning regarding enumerations that induced *Change Variable Type* and *Rename Variable* edits (Cloudstack #2833)

```
api/src/main/java/org/apache/cloudstack/api/comman
d/admin/diagnostics/RunDiagnosticsCmd.java
+     @Override
+     public String getEventType() {
+         VirtualMachine vm = _entityMgr.findById(Vi
+         if (vm.getType() == VirtualMachine.Type.Co
```

on 4 Sep 2018

it would be better to use a switch case instead of multiple if else, especially for comparing multiple enums

on 4 Sep 2018

Refactored to switch-case statements

4.4 How do Reviewers Suggest Refactorings in Refactoring-inducing PRs?

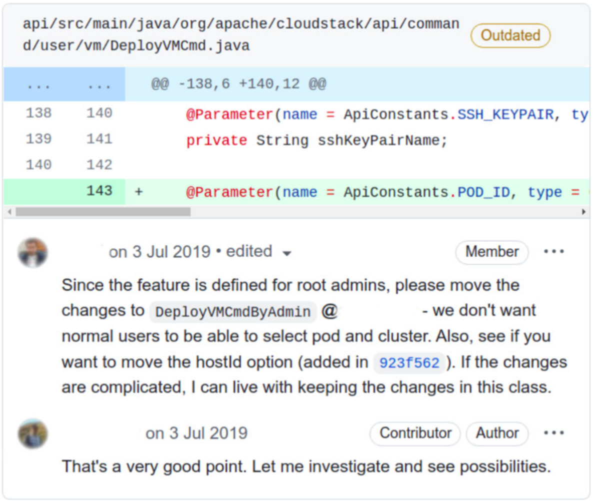
Refactoring suggestions are usually polite and precise, being well-understood by the PR authors in refactoring-inducing PRs. Those suggestions use expressions such as "***Maybe* ...** ", Kafka #5946 and "**...** ***can we*** **...**", Knox #74. We recognized this pattern, considering at least one review comment with polite and precise refactoring suggestions, in 23/30 PRs with refactoring edits exclusively induced by code review (Beam #4458, Brooklyn-Server #964 #1049, Cloudstack #2071 #2833 #3454, Dubbo #3174 #3299, Fluo #837, Kafka #4796 #5194 #5590 #5784 #6853 #7132, Knox #69 #74, Samza #1051, Servicecomb-Java-Chassis #346, Struts #43, Tika #234, Tinkerpop #1110, Tomee #407) and 11/14 PRs with refactorings induced by code review and other ones led by the PR authors (Dubbo #2279, Flink #7945 #7970 #8222, Incubator-Iceberg #119 #183, Kafka #4735 #5501 #5946 #6848, Rocketmq-Externals #45). We conjecture that the experience of reviewers against authors in refactoring-inducing PRs might explain such a pattern.

Sometimes, the reviewers provide a rationale for their suggestions, such as security-related issues in Cloudstack #3454 (Fig. 15). The purpose of a rationale is likely to let the author know that an adjustment is not necessary for the code operation but to bring more benefits. Review comments, in the form of warning on good practices of development, may induce refactoring and provide explanations and examples (using structures of the code under analysis, e.g., a class, a method) of how to deal with problematic points (e.g., single responsibility of methods in Incubator-Iceberg #119, Fig. 16).

Finding 7: Reviewers tend to make polite and precise suggestions for refactoring.

4.5 Do Suggestions of Refactoring Justify the Reasons?

Reviewers provide reasons that induced refactorings in 23/60 (38.3%) of the refactoring-inducing PRs. The refactoring type is not explicitly pointed, except for a few *Rename*, *Move*, and *Extract* instances, as it occurs in Cloudstack #2071 (Fig. 5). Suggestions of low-level refactorings may provide reasons, as in Kafka #5946 (Fig. 17). Only review comments in



```
api/src/main/java/org/apache/cloudstack/api/comman
d/user/vm/DeployVMCmd.java
@@ -138,6 +140,12 @@
      @Parameter(name = ApiConstants.SSH_KEYPAIR, ty
      private String sshKeyPairName;

+     @Parameter(name = ApiConstants.POD_ID, type =
```

on 3 Jul 2019 • edited

Since the feature is defined for root admins, please move the changes to `DeployVMCmdByAdmin` @ - we don't want normal users to be able to select pod and cluster. Also, see if you want to move the hostId option (added in `923f562`). If the changes are complicated, I can live with keeping the changes in this class.

on 3 Jul 2019

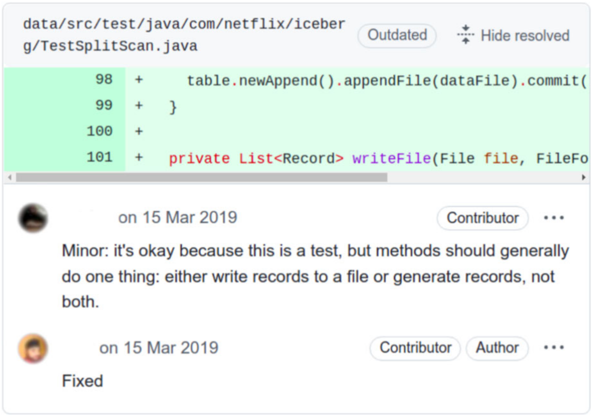
That's a very good point. Let me investigate and see possibilities.

Fig. 15 A review comment providing a reason (secure-related issues) that induced *Push Down Attribute* and *Push Down Method* edits (Cloudstack #3454)

Cloudstack #2071, Flink #7945, and Incubator-Iceberg #119 #183 submit reasons for high-level refactoring instances (Fig. 18).

Reasons are contextualized sentences regarding problematic situations, structured like alerts (e.g., constants format in Brooklyn-Server #1049, Fig. 13), sometimes accompanied by examples (e.g., code's structure under analysis in Kafka #5194). In other cases, reasons begin with a question on code logic (e.g., "*Is this type over-kill?*", "*Can we ... ?*") assisted by examples (e.g., questioning a class functionality in Flink #7945, Fig. 18). In other cases, reviewers provide a direct explanation (e.g., informing about a class useless in Dubbo #2279, Fig. 19). This result emphasizes code review as a traditional practice, conducted by expert reviewers, in Apache projects (Rigby et al. 2008). We identified no refactoring suggestion providing a reason for non-refactoring-inducing PRs.

In summary, we identified reasons in 13/30 refactoring-inducing PRs with refactorings exclusively induced by code review (Brooklyn-Server #964 #1049, Cloustack #2833 #3454, Dubbo #3174, Kafka #4796 #5784 #5590, Knox #74, Tika #234, Tinkerpop #1110, Tomee



```
data/src/test/java/com/netflix/iceber
g/TestSplitScan.java
+     table.newAppend().appendFile(dataFile).commit(
+   }
+
+   private List<Record> writeFile(File file, FileFo
```

on 15 Mar 2019

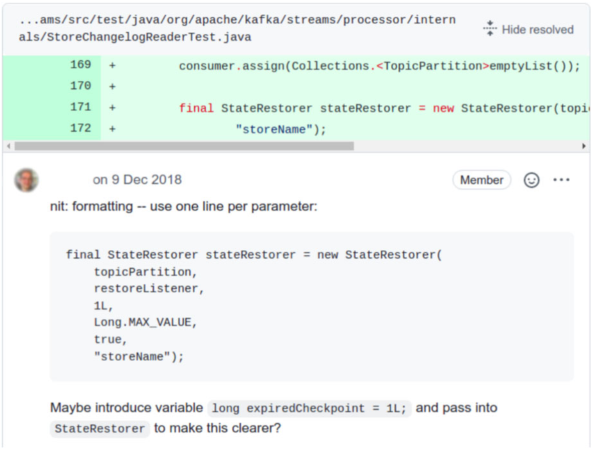
Minor: it's okay because this is a test, but methods should generally do one thing: either write records to a file or generate records, not both.

on 15 Mar 2019

Fixed

Fig. 16 A warning on responsibility of methods that induced *Change Return Type*, *Rename Method*, *Change Parameter Type*, and *Rename Parameter* edits (Incubator-Iceberg #119)

Fig. 17 A refactoring suggestion that induced an *Extract Variable* edit (Kafka #5946)



```
...ams/src/test/java/org/apache/kafka/streams/processor/internals/StoreChangelogReaderTest.java
+        consumer.assign(Collections.<TopicPartition>emptyList());
+
+        final StateRestorer stateRestorer = new StateRestorer(topi
+                "storeName");
```

on 9 Dec 2018

nit: formatting -- use one line per parameter:

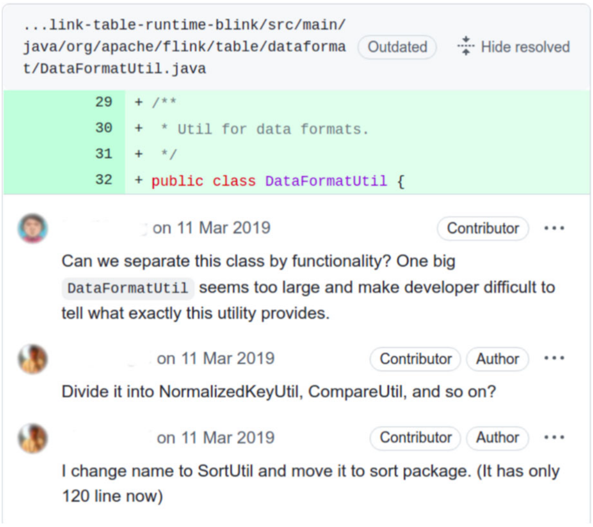
```
final StateRestorer stateRestorer = new StateRestorer(
    topicPartition,
    restoreListener,
    1L,
    Long.MAX_VALUE,
    true,
    "storeName");
```

Maybe introduce variable `long expiredCheckpoint = 1L;` and pass into `StateRestorer` to make this clearer?

#275 #407) and 8/14 PRs with refactorings led by the PR author and others suggested by code review (Avro #525, Dubbo #2279, Flink #8222, Incubator-Iceberg #119 #183, Kafka #5501 #5946 #6848). Reviewers provide examples in 1/30 PRs with refactorings uniquely induced by code review (Kafka #5194) and in 1/14 PRs with refactorings led by the PR author and others proposed by reviewers (Flink #7945).

Finding 8: Reviewers tend to provide reasons, occasionally supported by examples, to elucidate the benefits of refactoring.

Fig. 18 A review comment providing a reason (long class body) that induced *Move and Rename Class* edits (Flink #7945)



```
...link-table-runtime-blink/src/main/java/org/apache/flink/table/dataformat/DataFormatUtil.java
+ /**
+  * Util for data formats.
+  */
+ public class DataFormatUtil {
```

on 11 Mar 2019

Can we separate this class by functionality? One big `DataFormatUtil` seems too large and make developer difficult to tell what exactly this utility provides.

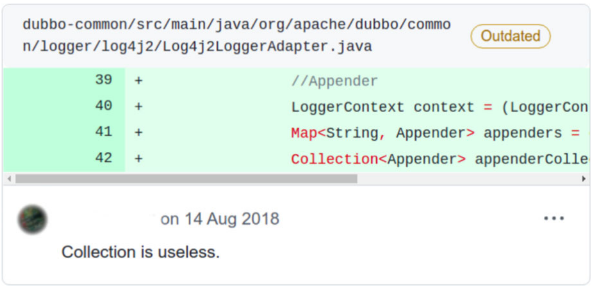
on 11 Mar 2019

Divide it into NormalizedKeyUtil, CompareUtil, and so on?

on 11 Mar 2019

I change name to SortUtil and move it to sort package. (It has only 120 line now)

Fig. 19 A warning on a class useless that induced *Change Variable Type* and *Rename Variable* edits (Dubbo #2279)



dubbo-common/src/main/java/org/apache/dubbo/common/logger/log4j2/Log4j2LoggerAdapter.java Outdated

```
+                //Appender
+                LoggerContext context = (LoggerCon
+                Map<String, Appender> appenders =
+                Collection<Appender> appenderColle
```

on 14 Aug 2018

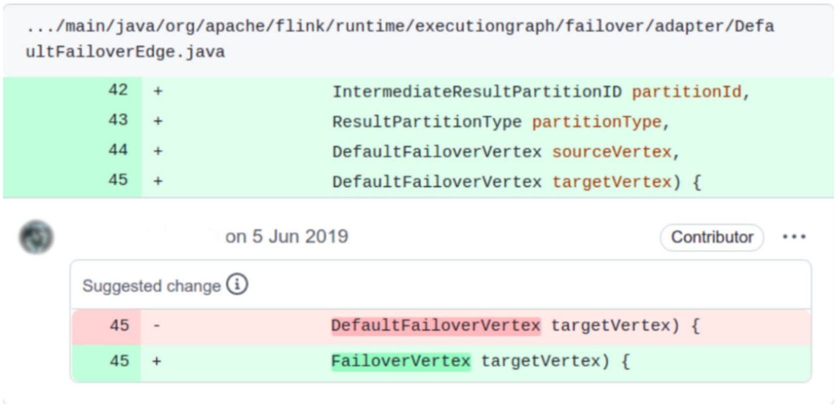
Collection is useless.

4.6 What is the Relationship Between Suggestions and Actual Refactorings in Refactoring-inducing PRs?

When the reviewers provide direct suggestions, refactoring edits are performed. We identify such a pattern in 44/44 (100%) of refactoring-inducing PRs, in which code review induced refactoring edits, that is, in 30/30 PRs with refactorings exclusively induced by reviewers and 14/14 PRs with refactorings led by the PR author and other ones suggested by code review. Direct suggestions may include explanations using examples in the code under analysis (e.g., suggesting a rename in Flink #8620, Fig. 20) and reasons (e.g., a large class body in Flink #7945, Fig. 18).

When there is space for discussion, refactorings may not be done. To reinforce such an emerged pattern, as we previously argued, the proportion of discussion is higher in non-refactoring-inducing PRs than in refactoring-inducing PRs. In particular, we identified a few discussions due to uncertain review comments (e.g., in Flink #9143, Fig. 7), and to impositions of change consisting of embedded code (e.g., a reviewer suggests a method change, using "*should we* ..." in Dubbo #4870, Fig. 21). In both cases, the authors presented arguments leading to no subsequent change.

Finding 9: Direct suggestions of refactoring tend to be embraced by the PR authors.



.../main/java/org/apache/flink/runtime/executiongraph/failover/adapter/DefaultFailoverEdge.java

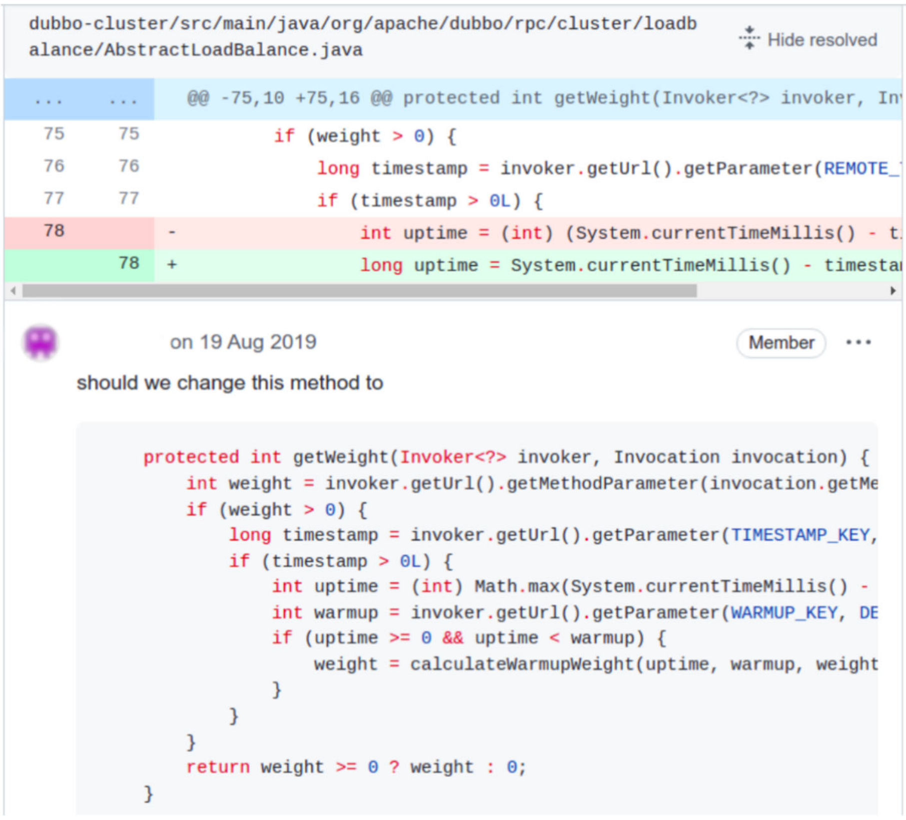
```
+            IntermediateResultPartitionID partitionId,
+            ResultPartitionType partitionType,
+            DefaultFailoverVertex sourceVertex,
+            DefaultFailoverVertex targetVertex) {
```

on 5 Jun 2019 Contributor

Suggested change

```
-            DefaultFailoverVertex targetVertex) {
+            FailoverVertex targetVertex) {
```

Fig. 20 A review comment that induced a *Rename Method* edit (Flink #8620)


dubbo-cluster/src/main/java/org/apache/dubbo/rpc/cluster/loadbalance/AbstractLoadBalance.java

```
@@ -75,10 +75,16 @@ protected int getWeight(Invoker<?> invoker, In
        if (weight > 0) {
            long timestamp = invoker.getUrl().getParameter(REMOTE_
            if (timestamp > 0L) {
-               int uptime = (int) (System.currentTimeMillis() - t
+               long uptime = System.currentTimeMillis() - timesta
```

on 19 Aug 2019

should we change this method to

```
protected int getWeight(Invoker<?> invoker, Invocation invocation) {
    int weight = invoker.getUrl().getMethodParameter(invocation.getMe
    if (weight > 0) {
        long timestamp = invoker.getUrl().getParameter(TIMESTAMP_KEY,
        if (timestamp > 0L) {
            int uptime = (int) Math.max(System.currentTimeMillis() -
            int warmup = invoker.getUrl().getParameter(WARMUP_KEY, DE
            if (uptime >= 0 && uptime < warmup) {
                weight = calculateWarmupWeight(uptime, warmup, weight
            }
        }
    }
    return weight >= 0 ? weight : 0;
}
```

Fig. 21 A review comment including an embedded code (Dubbo #4870)

4.7 General Considerations

From the scrutinization of a purposive sample of 118 Apache's merged PRs, we identified that (i) refactoring edits occur regardless of the change type (Finding 1) and (ii) the experience of the PR author (Finding 2) and the content of review comments (Findings 6 and 8) play a relevant role for refactoring-inducing PRs (Findings 3 – 5). Accordingly, review comments that induce refactorings submit crucial enhancements to the code and alert on good development practice, thus providing a learning time for less experienced PR authors and reviewers.

At a glance, our findings expand the knowledge on refactoring practice at the PR level since we discovered how authors and reviewers deal with refactoring opportunities in three distinct perspectives (refactoring-inducing PRs in which (i) code review induced all refactoring edits, (ii) the PRs' authors led all refactorings, and (iii) both code review and authors triggered refactoring edits, that is, both (i) and (ii) occurring simultaneously). Nevertheless, we realize the need for further investigation on the occurrence of specific refactoring types against the refactoring-inducement categories.

Findings 7 and 9 reveal a means to deal with a few challenges during code reviews (Bacchelli and Bird 2013; Ebert et al. 2021) when considering refactoring opportunities. In practice, even recognizing that reviewers have authority over the acceptance of PRs (Zanaty et al. 2018), our findings shed light on the need for review comments to be well-understood by the PR authors to achieve an effective (the proposed changes are eventually accepted) and efficient (the time this takes is as short as possible) code review (Rigby et al. 2015), in a scenario of drawbacks in communication among developers within PRs (Gousios et al. 2016) and claims for assisting reviewers in articulating appropriate comments (Bosu et al. 2017; Hossain et al. 2020; AlOmar et al. 2022). In this direction, review comments structured in a direct, polite, and precise manner can reduce the occurrence of conflicts and unnecessary discussion (Tsay et al. 2014; Chouchen et al. 2021; Ebert et al. 2021; Gonçalves et al. 2022), thus decreasing the PR review time (Kononenko et al. 2018). As a whole, our findings also emphasize that the reviewers' expertise is critical for a more productive code review, whereas enforcing the relevance of review comments well-structured for more effective communication at the PR level, thus corroborating with previous research (Kononenko et al. 2015; Gonçalves et al. 2023).

5 Implications and Guidelines

By characterizing code review in refactoring-inducing PRs, we can potentially advance the understanding of code reviewing at the PR level while assisting researchers, practitioners, tool builders, and educators in such a scenario.

Researchers: To the best of our knowledge, no prior MCR studies investigated PRs in light of our refactoring-inducing definition, thus exploring the refactorings performed specifically during the code review time (subsequent commits). In this context, Finding 1 sheds light on a novel view of works concerning pull-based development since researchers can consider our refactoring-inducing PR definition when designing studies on code review. For instance, as we identified 44/60 (73.3%) PRs encompassing at least one refactoring induced by code review, replications are welcome, but also investigations concerning missed refactoring opportunities in code review time and fault-properness in PRs with refactorings exclusively led by authors in contrast to ones with edits also induced by review comments.

Given Findings 2–6, we provide directions for researchers toward three dimensions. First, our study points out a few motivating factors behind refactoring-inducing PRs (Findings 2 and 6). Finding 2 is an expected result since we suppose that more experienced developers implement less problem-prone code. Nevertheless, Finding 6 indicates distinct characteristics of code reviewing in refactoring-inducing and non-refactoring-inducing PRs. Thus, reviewers of refactoring-inducing PRs are usually more experienced than their authors then such a pattern assists the knowledge transfer and learning of a less experienced author. This result corroborates Bacchelli and Bird that, when defining MCR, emphasize knowledge transfer as its relevant characteristic (Bacchelli and Bird 2013). However, the authors of non-refactoring-inducing PRs may miss opportunities to improve their code quality due to reviewer limitations (e.g., less experienced), although such a scenario provides a learning opportunity to a less experienced reviewer. This result reinforces other works on code review practice, such as Sadowski et al. which discovered education aspects as one of the main motivations behind code review at Google (Sadowski et al. 2018). Therefore, we recommend future research to explore strategies to support an effective code review participation of authors and reviewers, knowledge transfer, and awareness of the changes among team members at the PR level. In this sense, we suggest experimenting with requirements for reviewer recommendation systems in line with our findings, that is, considering the experience of the PR authors against the reviewers while monitoring the effectiveness and efficiency of code review. We conjecture results around this setup may provide valuable insights into the refactoring practice in code review time.

Second, Findings 3–5 reinforce the need for future (qualitative) research on MCR with PRs, distinguishing refactoring-inducing and non-refactoring-inducing PRs or considering their different characteristics when sampling PRs, as we claimed in our previous work (Coelho et al. 2021). Since we found patterns in the context of review comments, future studies may leverage supervised machine learning techniques to advance in understanding code review practice, by searching for these patterns in other GitHub repositories while checking for the presence of refactoring-inducement. We also conjecture that using embedded code in review comments may be understood, by PR authors, as a ready-made solution. Such a structure is not welcome by the authors; maybe, this could affect code authorship (in the developer's perception). For further investigation, the design of future studies may consider surveys with PR authors in order to validate our impressions of the embedded code's usefulness in code reviews.

Third, we strongly recommend reproduction studies intending to confirm or refute our findings by considering other GitHub projects and pull-based platforms. We provide a reproduction kit as support in this direction (Coelho 2024).

Practitioners: Based on our Findings 6–9, we provide a few guidelines for reviewers composing valuable review comments towards code refactoring in code review time:

- Be polite when suggesting refactorings, using expressions like "*can we ...?*", "*maybe ...*", as illustrated in Figs. 8, 9, 10, 14, 15, and 18;
- Be precise in both suggestions of refactoring and reasons (e.g., Figs. 10, 12, 14, and 19);
- Use the structure of the author's code (e.g., class names, method names, etc.) when providing explanations (e.g., Figs. 15 and 17). This usually makes more clear the expected improvements requested for the code author(s);
- Be direct when questioning the code logic, employing expressions that use the first person plural such as "*should we ...?*", as observed in Fig. 18;
- Avoid imposing any change, mainly by providing embedded code (Fig. 21 displays a counter-example).

Since code review is intrinsically a human task comprising technical, personal, and social aspects, following best practices may support developers in dealing with MCR challenges (Chouchen et al. 2021; Gonçalves et al. 2023). Those guidelines may improve reviewers' productivity by composing more effective review comments. Thus, reviewing discussions may be reduced. As a consequence, the time to merge PRs may also decrease, as argued by Gousios et al., because they found empirical evidence that code review affects the time to merge a PR (Gousios et al. 2014). We know that PR authors and reviewers are developers dealing with several tasks, including refactoring and reviewing code. In this setting, reviewers can incorporate our guidelines aiming at a more productive code review as getting away from conflicts/confusion/discussion scenarios while improving their capacity to politely communicate direct and precise refactoring suggestions to the code under review – this may be a familiar practice of expert developers but an unfamiliar one for the beginners. As a result, well-structured review comments also represent a rich learning opportunity for less experienced authors and reviewers.

Tool Builders: Supported by empirical evidence that integrating static analysis tools can improve the quality of code review (Balachandran 2013), we suggest the implementation of checkers of review comments as a feature available at code review boards. To emphasize the feasibility of this suggestion, Rahman et al. developed a machine learning prediction

Table 20 Validity and reliability countermeasures for characterizing code review in refactoring-inducing PRs

Type	Description
Internal validity	A design proposal to guide the study
Construct validity	Regular revision of the study design by supervisors
	Establishment of a chain of evidence based on data analyzed by three researchers (developers)
	Manual validation of refactoring edits in refactoring-inducing and non-refactoring-inducing PRs
Reliability	Effort towards clarifying the data analysis procedures to enable replications

model for assisting developers when formulating code review comments, based on the text and reviewer experience. Tool builders may leverage our guidelines in the sense of assisting more effective code review at the PR level since authors may miss opportunities to improve code due to not well-structured review comments. Although such a checker deals with natural language, we already identified a few terms and expressions to be avoided (e.g., "*not sure if* ...") and others that proved to be well accepted (e.g., "*How about ...?*"), which can be incorporated in comment checkers.

Concerning PR bots, the popular *Stale bot* has received a lot of criticism regarding its ineffectiveness in dealing with inactive PRs on GitHub (Khatoonabadi et al. 2023). Furthermore, custom-made bots, such as Apache flinkbot,[3] perform automated tasks, such as checking if a PR is changing the dependencies or the public API of the project. Therefore, there is a gap in PR bots helping with the actual code review process. We propose the development of code review bots aligned with our guidelines to assist reviewers and PR authors with refactoring-related issues. For instance, such bots could analyze the modified code in the PR and search for code reviewers who have more expertise on this part of the codebase and thus could suggest more relevant and appropriate refactorings.

Educators: Our dataset contains a large number of real code reviews from open-source projects that can be used as case studies in software engineering courses. The students can learn valuable lessons about how to give clear and appropriate feedback to a PR author, but also how to properly address the changes requested by a code reviewer. All cases in our dataset are well documented and include links to the corresponding PRs on GitHub.

6 Threats to Validity

To deal with the issues of **validity and reliability**, we propose the countermeasures introduced in Table 20. Because we consider data obtained in line with a specific data mining design, this study relies on its countermeasures and threats to validity and reliability described in our previous work (Coelho et al. 2021). It is noteworthy that we elaborated the research design supported by guidelines (Creswell 2012), whereas we employed special attention for sampling representative PRs in each round of analysis. Accordingly, each purposive sample comprises one refactoring-inducing PR from the diversity of settings available at that

[3] https://github.com/apache/flink/pull/6873

moment. For instance, the 13 refactoring-inducing PRs of sample 3 (Round 3) represent a set of available compositions of high-level refactorings found in 36 refactoring-inducing PRs, having high-level refactorings, at the beginning of the round. For each round of analyses, we used randomness to select non-refactoring-inducing PRs, having an equal median of the number of review comments in the set of refactoring-inducing PRs, intending a fair comparison for answering RQ_1 and RQ_2. We considered the median value because review comments present a non-normal distribution in our whole sample (Coelho 2024). Even so, there are risks of threats to internal validity due to any non-previously identified deficiencies in our research design.

We tried to establish a chain of evidence for the data interpretation and systematically explain the procedures, decisions, and obtained results in each design step. We also propose a few actions intending to increase validity, including a sanity check of refactoring edits (checking false positives and false negatives) and a data analysis performed by three researchers intending to reduce researcher bias.

The subjective nature of our qualitative study does not allow generalizations. However, we speculate that we can extend our findings to cases that have common characteristics with Apache's projects, that is other OSS projects that follow a geographically distributed development (Zhong et al. 2012) and are aligned to "the Apache way" principles (Apa 2019b). We systematically structured all procedures to deal with reliability issues, thus providing a reproduction kit to enable replications, publicly available at (Coelho 2024).

7 Related Work

The code review practice has been the focus of characterization studies that explore both OSS and industrial scenarios. Hence, it is perceptible the enhancement of techniques, tools, and recommendations for supporting the code review as a result of research initiatives over time.

The first studies aimed to understand the differences between software inspection as performed in industry and peer code review in OSS development. Rigby et al., for this purpose, provide theory and study methodology in light of metrics similar to those considered in software inspection (Rigby et al. 2008). They characterized Apache's peer reviews as early, frequent reviews of small, independent, complete contributions conducted asynchronously by a small number of expert reviewers. *These findings express code reviewing as a traditional practice in Apache projects, which inspired us to explore refactoring-inducing PRs in their repositories.*

The differential of OSS code review practices for code quality has become evident, as claimed by Rigby et al. when suggesting a few recommendations suitable to the industry from the OSS code reviewing (Rigby et al. 2012). In this context, Rigby and Bird studied convergent code review practices in both industrial and OSS scenarios aiming at discovering efficient methods of review (Rigby and Bird 2013). Also, Rigby et al. extended the knowledge on peer review practice, examining many aspects, including the experience (i.e., length of time a developer has been with a project) of authors and reviewers (Rigby et al. 2014). They found the reviewers typically have more experience than authors. *We expand the knowledge about the experience of Apache authors and reviewers by providing arguments denoting a difference between the experience of authors and reviewers in refactoring-inducing and non-refactoring-inducing PRs.*

We noticed a series of research initiatives to better understand MCR. Sadowski et al. studied code review practice at Google, discovering educational aspects as one of its motivations (Sadowski et al. 2018). As well, *our findings reinforce the intrinsic nature of code review in terms of learning opportunities*. McIntosh et al. explored the relationship between MCR and software quality, identifying that a low proportion of changes reviewed and the involvement degree of reviewers generate additional post-release defects (McIntosh et al. 2014, 2016). Beller et al. investigated the benefits of MCR by examining the issues fixed at code review time in OSS projects. They found that most of the changes are due to review comments, concerning code improvement instead of fixing defects (Beller et al. 2014). *As a whole, those findings reinforce code improvement as a relevant objective of MCR, thus motivating us to advance knowledge on code reviewing-related aspects such as review comments.*

Also, studies explored code review quality. Kononenko et al. found empirical evidence that aspects such as reviewer experience (i.e., the overall number of completed reviews) and the thoroughness of feedback are associated with the code review quality (Kononenko et al. 2015, 2016). Bosu et al. identified factors influencing the review comments' usefulness (Bosu et al. 2015) – for instance, review comments asking questions to understand code were considered non-useful. However, such an aspect may induce refactoring edits at the PR level in line with our findings. *In this context, our guidelines for structuring review comments denote a differential of our study.*

Pull-based development is the focus of a few empirical studies. Gousios et al. investigated GitHub PRs data to understand the factors influencing their effectiveness (e.g., code churn, number of commits, and number of review comments) (Gousios et al. 2014). In particular, besides providing us with a few features for exploring in our studies, this work found empirical evidence that code review affects the time to merge a PR. *We advanced the knowledge regarding pull-based development when we found that refactoring-inducing PRs with refactorings induced by code review take less time to merge than refactoring-inducing PRs with refactorings led by PR authors.*

Review comments in pull-based development have been examined in a few studies. Rahman et al. developed a machine learning prediction model for assisting developers when formulating code review comments in light of their usefulness, based on the content of review comments and reviewer experience (Rahman et al. 2017). This work emphasizes the reviewer experience as a relevant factor influencing the useful review comments, whereas *our studies shed light on considering the experience of reviewers against authors when dealing with refactorings at the PR level*. Li et al. found 15 typical code review patterns in GitHub PRs such as reviewing for code correctness (e.g., indicating blank lines) (Li et al. 2017). *We go further in the sense of encompassing code review properties while investigating the refactoring-inducing and non-refactoring-inducing PRs*. Bosu et al. revisited the OSS and industrial scenarios to investigate code review practice, including interactivity aspects and human effort required in review-related tasks (Bosu et al. 2017). They identified a few subjects for further research, such as assisting reviewers in articulating review comments. In particular, *our guidelines are an initiative to meet such a claim.*

Distinct from the previous works focusing on the commit level, Pantiuchina et al. observed that code readability, change- and fault-proneness, and experience of developers are factors influencing refactorings when exploring the PR level (Pantiuchina et al. 2020). They analyzed the discussion and commits of merged PRs, containing at least one refactoring in any one of their commits, without pre-processing squashed and merged PRs (to the best of our knowledge). They found that most of the refactorings are triggered by either the original intents of PRs or discussions with other developers. Their findings are motivating, since they indicate the influence that code review has on refactoring edits at the PR level. Our studies differ from such previous work because *we propose an exploration of refactorings performed as part of changes at subsequent commits from merged PRs, when investigating code reviewing-related aspects and refactoring-inducement*.

Paixão et al. investigated intentions behind refactorings performed during code reviewing, by analyzing Gerrit reviews, and found that motivations may emerge from code reviews and, in turn, influence the composition of edits and number of reviews (Paixão et al. 2020). Their findings suggest aspects of refactoring and code review to be explored at the GitHub PR level (e.g., typical types of refactoring edits). Specifically, they answered research questions based on the evolution of refactorings at code reviewing time in light of distinct types of developers' intents. In addition, Panichella and Zaugg proposed a taxonomy for types of review changes, including refactoring as a structural change (Panichella and Zaugg 2020). *Our study scrutinizes refactoring-related aspects according to the refactoring-inducement context, independently of developers' intents*.

In a nutshell, our main goal is to fill up the knowledge gap on PRs aligned to our refactoring-inducing definition instead of investigating motivations behind refactoring edits at the code review time. For that, we first examined the differences between refactoring-inducing and non-refactoring-inducing PRs (Coelho et al. 2021). Accordingly, we speculate that our findings complement the related studies in the sense of a better understanding of MCR practice concerning code evolution.

8 Concluding Remarks

In this qualitative study, we explored 118 PRs, the experience of their reviewers, 923 review comments, and 361 refactoring edits aiming at characterizing code review in refactoring-inducing PRs. Our results reveal motivating factors behind refactoring-inducing PRs, technical aspects of the structure of review comments, and guidelines for a more productive code review. Thus, we found the PR author's experience and the structure of review comments are two reasons behind refactoring-inducing PRs – specifically, review comments are polite and precise while addressing major issues on code logic, improvements (including refactorings directly), and warnings on good development practices.

MCR and code refactoring are complex processes dealing with technical and non-technical challenges. Thus, we believe that understanding code review in the presence of refactoring is relevant as a baseline to identify improvement opportunities for pull-based software development. Accordingly, we designed a qualitative investigation of a software ecosystem (Apache) and concentrated effort into building guidance for a more productive code review, besides suggesting directions for researchers, practitioners, tool builders, and educators.

As future research, we propose (i) scrutinizing the refactoring types in our dataset to gain an in-depth understanding of refactoring practice in refactoring-inducing PRs with edits exclusively led by the authors against those triggered by code review; for instance, identifying whether there are refactoring types more acceptable than others when suggested by reviewers; (ii) designing surveys with developers to enhance our conclusions to confirm/refute our guidelines for more effective review comments; (iii) assessing the guidelines for valuable review comments towards code refactoring by running a controlled experiment with a group of reviewers and observe both the quality of review comments and code changes during reviewing; and (iv) confirming/refuting our findings, by replication, in other projects in both OSS and industrial scenarios.

Acknowledgements We thank the anonymous reviewers for their suggestions to improve this manuscript; and Hugo Addobbati and Ramon Fragoso for their valuable contributions to the qualitative data analysis. This research was partially supported by the National Council for Scientific and Technological Development (CNPq)/Brazil (process 429250/2018-5).

Data Availibility The data that support the findings of this study are openly available at https://github.com/flaviacoelho/emse-2024-reproduction-kit.

References

(2001) Manifesto for Agile software development. https://agilemanifesto.org/, accessed on: August 2020

(2018) RefactoringMiner – a refactoring detection tool. https://github.com/tsantalis/RefactoringMiner, accessed on: September 2019

(2019a) The Apache® Software Foundation expands infrastructure with GitHub integration. https://blogs.apache.org/foundation/entry/the-apache-software-foundation-expands, accessed on: June 2020

(2019b) Briefing: The Apache way. https://www.apache.org/theapacheway/, accessed on: June 2020

(2020) The Apache® Software Foundation projects statistics. https://projects.apache.org/statistics.html, accessed on: December 2020

(2021) Community-led development "The Apache Way". https://apache.org/foundation/how-it-works.html#roles, accessed on: June 2021

(2022) GitHub pull requests. https://help.github.com/en/github/collaborating-with-issues-and-pull-requests, accessed on: August 2022

AlOmar E, Mkaouer M, Ouni A (2019) Can refactoring be self-affirmed? An exploratory study on how developers document their refactoring activities in commit messages. In: Proc. Int. Ws. Refactoring, Montreal, Canada, pp 51–58, https://doi.org/10.1109/IWoR.2019.00017

AlOmar E, AlRubaye H, Mkaouer M, et al (2021) Refactoring practices in the context of modern code review: An industrial case study at Xerox. In: Proc. IEEE/ACM Int. Conf. Softw. Eng. – Softw. Eng. in Practice, Virtual, pp 348–357, https://doi.org/10.1109/ICSE-SEIP52600.2021.00044

AlOmar E, Chouchen M, Mkaouer M, Ouni A (2022) Code review practices for refactoring changes: An empirical study on openstack. In: Proc. Int. Conf. Mining Softw. Repositories, pp 689–701, https://doi.org/10.1145/3524842.3527932

Alves E, Song M, Massoni T et al (2018) Refactoring inspection support for manual refactoring edits. IEEE Trans Softw Eng 44(4):365–383. https://doi.org/10.1109/TSE.2017.2679742

Bacchelli A, Bird C (2013) Expectations, outcomes, and challenges of modern code review. Proc. Int. Conf. Softw. Eng, San Francisco, USA, pp 712–721

Balachandran V (2013) Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. Proc. Int. Conf. Softw. Eng, San Francisco, USA, pp 931–940

Baysal O, Kononenko O, Holmes R, Godfrey M (2016) Investigating technical and non-technical factors influencing modern code review. Empirical Softw Eng 21(3):932–959. https://doi.org/10.1007/s10664-015-9366-8

Beller M, Bacchelli A, Zaidman A, Juergens E (2014) Modern code reviews in open-source projects: Which problems do they fix? In: Proc. Working Conf. Mining Softw. Repositories, Hyderabad, India, pp 202–211, https://doi.org/10.1145/2597073.2597082

Bosu A, Greiler M, Bird C (2015) Characteristics of useful code reviews: An empirical study at Microsoft. Proc. Working Conf. Mining Softw. Repositories, Florence, Italy, pp 146–156

Bosu A, Carver J, Bird C et al (2017) Process aspects and social dynamics of contemporary code review: Insights from open source development and industrial practice at Microsoft. IEEE Trans Softw Eng 43(1):56–75. https://doi.org/10.1109/TSE.2016.2576451

Brown C, Parnin C (2020) Understanding the impact of github suggested changes on recommendations between developers. In: Proc. ACM Joint Meeting European Softw. Eng. Conf. Symp. Foundations Softw. Eng., Virtual Event, USA, pp 1065–1076. https://doi.org/10.1145/3368089.3409722

Burdess N (2010) Starting statistics: a short, clear guide. Sage Publications

Chacon S, Straub B (2014) Pro Git, 2nd edn. Apress, New York, USA

Chouchen M, Ouni A, Kula RG, et al (2021) Anti-patterns in modern code review: Symptoms and prevalence. In: Proc. Int. Conf. Softw. Analysis, Evol. Reengineering, Virtual, pp 531–535. https://doi.org/10.1109/SANER50967.2021.00060

Coelho F (2022) Characterizing refactoring-inducing pull requests. PhD thesis, Acad. Unit Comp. Sci., Fed. Univ. Campina Grande, Brazil

Coelho F (2024) A qualitative study on refactorings induced by code review – Reproduction kit. https://github.com/flaviacoelho/emse-2024-reproduction-kit

Coelho F, Tsantalis N, Massoni T, Alves E (2021) An empirical study on refactoring-inducing pull requests. In: Proc. Int. Symp. Empir. Softw. Eng. Meas., Bari, Italy, pp 1–12. https://doi.org/10.1145/3475716.3475785

Creswell J (2012) Qualitative Inquiry and Research Design: Choosing among Five Traditions, 3rd edn. Sage Publications, Thousand Oaks, EUA

Dohmke T (2023) 100 million developers and counting. https://github.blog/2023-01-25-100-million-developers-and-counting/, accessed on: January 2024

Ebert F, Castor F, Novielli N, Serebrenik A (2021) An exploratory study on confusion in code reviews. Empir Softw Eng 26(1):1–48. https://doi.org/10.1007/s10664-020-09909-5

Fowler M (1999) Refactoring: Improving the Design of Existing Code, 1st edn. Addison-Wesley Longman Publishing, Westford, USA

Fowler M (2018) Refactoring: Improving the Design of Existing Code, 2nd edn. Addison-Wesley Professional

Gonçalves P, Çalikli G, Bacchelli A (2022) Interpersonal conflicts during code review: Developers' experience and practices. Proc ACM Hum-Comput Interact 6(CSCW1), https://doi.org/10.1145/3512945

Gonçalves P, Calikli G, Serebrenik A, Bacchelli A (2023) Competencies for code review. Proc ACM Hum-Comput Interact 7(CSCW1), https://doi.org/10.1145/3579471

Gousios G, Pinzger M, Av Deursen (2014) An exploratory study of the pull-based software development model. Proc. Int. Conf. Softw. Eng, Hyderabad, India, pp 345–355

Gousios G, Storey MA, Bacchelli A (2016) Work practices and challenges in pull-based development: The contributor's perspective. In: Proceedings of the 38th International Conference on Software Engineering, ACM, Austin, USA, ICSE '16, pp 285–296, https://doi.org/10.1145/2884781.2884826

Hossain S et al (2020) Measuring the effectiveness of software code review comments. In: Singh M, Gupta PK, Tyagi V, Flusser J, Ören T, Valentino G (eds) Proc. Advances in Comput. and Data Sci, Springer Singapore, Singapore, pp 247–257

Humble J, Farley D (2010) Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, 1st edn. Addison-Wesley Professional

Khatoonabadi S, Costa D, Mujahid S, Shihab E (2023) Understanding the helpfulness of stale bot for pull-based development: An empirical study of 20 large open-source projects. ACM Trans Softw Eng Methodol 33(2), https://doi.org/10.1145/3624739

Kononenko O, Baysal O, Guerrouj L, Cao Y, Godfrey M (2015) Investigating code review quality: Do people and participation matter? In: Proc. IEEE Int. Conf. Softw. Maintenance Evol., Bremen, Germany, pp 111–120, https://doi.org/10.1109/ICSM.2015.7332457

Kononenko O, Baysal O, Godfrey M (2016) Code review quality: How developers see it. In: Proc. Int. Conf. Softw. Eng., Austin, EUA, pp 1028–1038, https://doi.org/10.1145/2884781.2884840

Kononenko O, Rose T, Baysal O, et al (2018) Studying pull request merges: A case study of Shopify's active merchant. In: Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice, ACM, Gothenburg, Sweden, ICSE-SEIP '18, pp 124–133, https://doi.org/10.1145/3183519.3183542

Li ZX, Yu Y, Yin G, Wang T, Wang HM (2017) What are they talking about? Analyzing code reviews in pull-based development model. J Comput Sci Technol 32(6):1060–1075. https://doi.org/10.1007/s11390-017-1783-2

MacLeod L, Greiler M, Storey MA, Bird C, Czerwonka J (2018) Code reviewing in the trenches: Challenges and best practices. IEEE Software 35(04):34–42. https://doi.org/10.1109/MS.2017.265100500

McIntosh S, Kamei Y, Adams B, Hassan A (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: Proc. Working Conf. Mining Softw. Repositories, Hyderabad, India, pp 192–201, https://doi.org/10.1145/2597073.2597076

McIntosh S, Kamei Y, Adams B, Hassan A (2016) An empirical study of the impact of modern code review practices on software quality. Empir Softw Eng 21(5):2146–2189. https://doi.org/10.1007/s10664-015-9381-9

Mockus A, Votta L (2000) Identifying reasons for software changes using historic databases. In: Proc. Int. Conf. Softw. Maint., Washington, USA, pp 120–130, https://doi.org/10.1109/ICSM.2000.883028

Opdyke W, Johnson R (1990) Refactoring: An aid in designing application frameworks and evolving object-oriented systems. In: Proc. Symp. Obj. Orien. Prg. Emph. Pract. Apps

Paixão M, Maia P (2019) Rebasing in code review considered harmful: A large-scale empirical investigation. Proc. Int. Work. Conf. Source Code Analys. and Manip, Cleveland, USA, pp 45–55

Paixão M, Uchôa A, Bibiano A et al (2020) Behind the intents: An in-depth empirical study on software refactoring in modern code review. Proc. Int. Conf. Mining Softw. Repositories, Virtual, pp 125–136

Palomba F, Zaidman A, Oliveto R, De Lucia A (2017) An exploratory study on the relationship between changes and refactoring. In: Proc. Int. Conf. on Prg. Comprehension, Buenos Aires, Argentina, pp 176–185, https://doi.org/10.1109/ICPC.2017.38

Panichella S, Zaugg N (2020) An empirical investigation of relevant changes and automation needs in modern code review. Empir Softw Eng 25:4833–4872. https://doi.org/10.1007/s10664-020-09870-3

Pantiuchina J, Zampetti F, Scalabrino S et al (2020) Why developers refactor source code: A mining-based study. ACM Trans Softw Eng Methodol 29(4):1–30. https://doi.org/10.1145/3408302

Patton M (2014) Qualitative Research & Evaluation Methods: Integrating Theory and Practice, 4th edn. Sage Publications, Thousand Oaks, EUA

Rahman M, Roy C, Kula R (2017) Predicting usefulness of code review comments using textual features and developer experience. In: Proc. Int. Conf. Mining Softw. Repositories, Buenos Aires, Argentina, pp 215–226, https://doi.org/10.1109/MSR.2017.17

Rigby P, Bird C (2013) Convergent contemporary software peer review practices. In: Proc. Joint Meeting Foundations Softw. Eng., Saint Petersburg, Russia, pp 202–212, https://doi.org/10.1145/2491411.2491444

Rigby P, German D, Storey MA (2008) Open source software peer review practices: A case study of the apache server. In: Proc. Int. Conf. Softw. Eng., Leipzig, Germany, pp 541–550, https://doi.org/10.1145/1368088.1368162

Rigby P, Cleary B, Painchaud F et al (2012) Contemporary peer review in action: Lessons from open source development. IEEE Softw 29(6):56–61. https://doi.org/10.1109/MS.2012.24

Rigby P, German D, Cowen L, Storey MA (2014) Peer review on open-source software projects: Parameters, statistical models, and theory. ACM Trans Softw Eng Methodol 23(4):1–33. https://doi.org/10.1145/2594458

Rigby P, Bacchelli A, Gousios G, Mukadam M (2015) Chapter 9 - a mixed methods approach to mining code review data: Examples and a study of multicommit reviews and pull requests. In: Bird C, Menzies T, Zimmermann T (eds) The Art and Science of Analyzing Software Data. Morgan Kaufmann, Boston, pp 231–255

Sadowski C, Söderberg E, Church L, et al. (2018) Modern code review: A case study at Google. In: Proc. Int. Conf. Softw. Eng.: Softw. Eng. Prac., Gothenburg, Sweden, pp 181–190, https://doi.org/10.1145/3183519.3183525

Silva D, Tsantalis N, Valente M (2016) Why we refactor? confessions of GitHub contributors. In: Proc. Proc. ACM SIGSOFT Int. Symp. Found. Softw. Eng., pp 858–870, https://doi.org/10.1145/2950290.2950305

Swanson E (1976) The dimensions of maintenance. Proc. Int. Conf. Softw. Eng, San Francisco, USA, pp 492–497

Szőke G, Nagy C, Ferenc R, Gyimóthy T (2014) A case study of refactoring large-scale industrial systems to efficiently improve source code quality. Proc. Int. Conf. Comput. Sci. Apps, Guimarães, Portugal, pp 524–540

Thongtanunam P, McIntosh S, Hassan A, Iida H (2016) Revisiting code ownership and its relationship with software quality in the scope of modern code review. In: Proc. Int. Conf. Softw. Eng., Austin, Texas, pp 1039—-1050, https://doi.org/10.1145/2884781.2884852

Tsantalis N, Mansouri M, Eshkevari L, et al. (2018) Accurate and efficient refactoring detection in commit history. In: Proceedings of the 40th International Conference on Software Engineering, ACM, Gothenburg, Sweden, ICSE '18, pp 483–494, https://doi.org/10.1145/3180155.3180206

Tsantalis N, Ketkar A, Dig D (2020) RefactoringMiner 2.0. IEEE Transactions on Software Engineering pp 1–1, https://doi.org/10.1109/TSE.2020.3007722, early access article

Tsay J, Dabbish L, Herbsleb J (2014) Let's talk about it: Evaluating contributions through discussion in GitHub. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, Hong Kong, China, FSE '14, pp 144–154, https://doi.org/10.1145/2635868.2635882

Uchôa A, Barbosa C, Oizumi W et al (2020) How does modern code review impact software design degradation? an in-depth empirical study. Proc. IEEE Int. Conf. Softw. Maintenance Evol, Adelaide, Australia, pp 511–522

Vassallo C, Grano G, Palomba F et al (2019) A large-scale empirical exploration on refactoring activities in open source software projects. Sci Comput Program 180:1–15

Wang S, Bansal C, Nagappan N, Philip A (2019) Leveraging change intents for characterizing and identifying large-review-effort changes. In: Proc. Int. Conf. Predictive Models and Data Analytics in Softw. Eng., Recife, Brazil, pp 46–55, https://doi.org/10.1145/3345629.3345635

Zanaty F, Hirao T, McIntosh S, Ihara A, Matsumoto K (2018) An empirical study of design discussions in code review. In: Proc. Int. Symp. Empir. Softw. Eng. Meas., Oulu, Finland, https://doi.org/10.1145/3239235.3239525

Zhong H, Yang Y, Keung J (2012) Assessing the representativeness of open source projects in empirical software engineering studies. Proc. APac. Softw. Eng. Conf, Hong Kong, China, pp 808–817

Zhu J, Zhou M, Mockus A (2016) Effectiveness of code contribution: From patch-based to pull-request-based tools. In: Proc. ACM SIGSOFT Int. Symp. Found. Softw. Eng., Seattle, USA, pp 871–882, https://doi.org/10.1145/2950290.2950364