An Empirical Study on Refactoring-Inducing Pull Requests

Flávia Coelho
Federal University of Campina Grande
Campina Grande, Brazil
flavia@copin.ufcg.edu.br

Nikolaos Tsantalis
Concordia University
Montreal, Canada
nikolaos.tsantalis@concordia.ca

Tiago Massoni
Federal University of Campina Grande
Campina Grande, Brazil
massoni@computacao.ufcg.edu.br

Everton L. G. Alves
Federal University of Campina Grande
Campina Grande, Brazil
everton@computacao.ufcg.edu.br

ABSTRACT

Background: Pull-based development has shaped the practice of Modern Code Review (MCR), in which reviewers can contribute code improvements, such as refactorings, through comments and commits in Pull Requests (PRs). Past MCR studies uniformly treat all PRs, regardless of whether they induce refactoring or not. We define a PR as *refactoring-inducing*, when refactoring edits are performed after the initial commit(s), as either a result of discussion among reviewers or spontaneous actions carried out by the PR developer. **Aims:** This mixed study (quantitative and qualitative) explores code reviewing-related aspects intending to characterize refactoring-inducing PRs. **Method:** We hypothesize that refactoring-inducing PRs have distinct characteristics than non-refactoring-inducing ones and thus deserve special attention and treatment from researchers, practitioners, and tool builders. To investigate our hypothesis, we mined a sample of 1,845 Apache's merged PRs from GitHub, mined refactoring edits in these PRs, and ran a comparative study between refactoring-inducing and non-refactoring-inducing PRs. We also manually examined 2,096 review comments and 1,891 detected refactorings from 228 refactoring-inducing PRs. **Results:** We found 30.2% of refactoring-inducing PRs in our sample and that they significantly differ from non-refactoring-inducing ones in terms of number of commits, code churn, number of file changes, number of review comments, length of discussion, and time to merge. However, we found no statistical evidence that the number of reviewers is related to refactoring-inducement. Our qualitative analysis revealed that at least one refactoring edit was induced by review in 133 (58.3%) of the refactoring-inducing PRs examined. **Conclusions:** Our findings suggest directions for researchers, practitioners, and tool builders to improve practices around pull-based code review.

CCS CONCEPTS

• **Software and its engineering** → *Programming teams*; **Software evolution**.

KEYWORDS

refactoring-inducing pull request, code review mining, empirical study

ACM Reference Format:
Flávia Coelho, Nikolaos Tsantalis, Tiago Massoni, and Everton L. G. Alves. 2021. An Empirical Study on Refactoring-Inducing Pull Requests. In *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (ESEM '21), October 11–15, 2021, Bari, Italy.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3475716.3475785

1 INTRODUCTION

In *Modern Code Review* (MCR), developers review code changes in a lightweight, tool-assisted, and asynchronous manner [18]. In this context, regular change-based reviewing, in which code improvements are embraced, became an essential practice in the MCR scenario [18, 66]. Code changes may comprise new features, bug fixes, or other maintenance tasks, providing potential opportunities for refactorings [60], which in turn form a significant part of the changes [19, 75]. Empirical evidence suggests a distinction between refactoring-dominant changes and other types. For instance, reviewing bug fixes is more time-consuming than reviewing refactorings, since the latter preserve code behavior [69]. Given the nature of changes significantly affects code review effectiveness [63], as it directly influences how reviewers perceive the changes, the provision of suitable resources for assisting code review is essential.

Characterization studies of MCR have been conducted to investigate technical aspects of reviewing [20, 24, 41, 66–68, 71], factors leading to useful code review [25], circumstances that contribute to code review quality [45], and general code review patterns in pull-based development [49]. Those studies are relevant because MCR is critical in repository-based software development, especially in *Agile software development*, driven by change and collaboration [1].

In practice, Git *Pull Requests* (PRs) are relevant to MCR as they promote well-defined and collaborative reviewing. Through PRs, the code is subject to a review process in which reviewers may suggest improvements before merging the code to the main branch of a repository [29]. Such improvements may take the form of refactorings, resulting from discussions among the PR author and reviewers on code quality issues, including spontaneous actions of the PR author aiming to refine the originally submitted solution. We hypothesize that PRs that induce refactoring edits have different characteristics from those that do not, as refactoring may involve design and API changes that require more extensive effort, discussion and knowledge of the project. It is worth clarifying that

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ESEM '21, October 11–15, 2021, Bari, Italy
© 2021 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-8665-4/21/10…$15.00
https://doi.org/10.1145/3475716.3475785

this study sheds light on refactorings induced by code review (Section 4) aiming to provide an initial understanding of how review discussions induce such edits.

Motivation: By distinguishing refactoring-inducing from non-refactoring-inducing PRs, we can potentially advance the understanding of code reviewing at the PR level and assist researchers, practitioners, and tool builders in this context. No prior MCR studies made a distinction between refactoring-inducing and non-refactoring-inducing PRs, when analyzing their research questions, which might have affected their findings or discussions. For instance, by also regarding refactoring-inducing PRs, Gousios et al. [37] and Kononenko et al. [46] could have found different factors influencing the time to merge a PR; Li et al. [49] could have included refactoring concerns to the multilevel taxonomy for review comments in the pull-based development model; Pascarella et al. [62] could have identified further information to perform a proper code review in presence of refactorings; Paixão et al. [17] could have complemented the study on the reasons for refactorings during code review when analyzing projects in Gerrit; whereas, Pantiuchina et al. [61] could have different conclusions on the motivations for refactorings in PRs, since they analyzed PRs in which refactorings were detected even in the initial commit (i.e., these refactorings were not induced from reviewer discussions). In practice, being unaware of refactoring-inducing PRs' characteristics, practitioners and tool builders might miss opportunities to manage better their resources and to assist developers in PRs, respectively. Moreover, a refactoring-aware notification system could help in allocating reviewers with more knowledge on the design of the refactored code when a PR becomes refactoring-inducing, as design changes caused by refactoring need to be more extensively discussed and agreed upon.

Definition 1.1. 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 contributor. 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 pull requests $P(u_q) = \{p_1, p_2, ..., p_m\}$ over time. Each pull request 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, $I(p_j) \subseteq C(p_j)$. A refactoring-inducing pull request is that in which $\exists\ c_k \mid R(c_k) \neq \emptyset$, where $R(c_k)$ denotes the set of refactorings performed in commit c_k and $|I(p_j)| < k \leq n$.

To clarify our definition, Figure 1 depicts a refactoring-inducing PR consisting of three initial commits ($c_1 - c_3$) and six subsequent commits ($c_4 - c_9$), three of which include refactoring edits (c_5, c_7, c_8), e.g., commit c_7 has two *Rename Class* and three *Change Variable Type* refactoring instances. Our study explores differences/similarities between PRs based on the refactorings performed in PR commits subsequent to the initial ones ($c_4 - c_9$).

We propose an investigation at the PR level because we understand it as a complete scenario for exploring code reviewing practices in a well-defined scope of development, which allows us to go beyond an investigation at the commit level. For instance, we can obtain a global comprehension of contributions to the original code, in terms of both commits and reviewing-related aspects (e.g., reviewers' comments). Our conception is mainly inspired by empirical evidence showing that pull-based development is associated with larger numbers of contributions [81], and that PR discussions lead to additional refactorings [61]. To guide our investigation, we designed the following research questions:

- RQ$_1$: How common are refactoring-inducing PRs?
- RQ$_2$: How do refactoring-inducing PRs compare to non-refactoring-inducing ones?
- RQ$_3$: Are refactoring edits induced by code reviews?

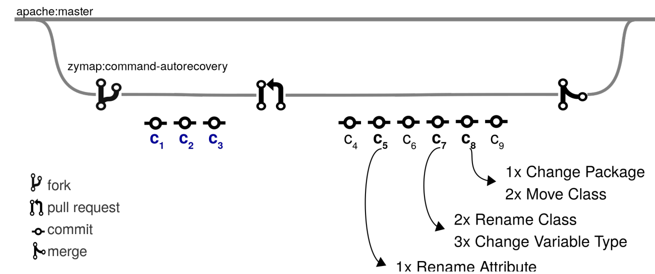


Figure 1: A Refactoring-Inducing Pull Request (Apache bookkeeper PR #2010), Illustrating Initial Commits ($c_1 - c_3$) and Subsequent Commits ($c_4 - c_9$).

We mined merged PRs from Apache's Java repositories in GitHub, and we used state-of-the-art tools and techniques, such as RefactoringMiner [11] and *Association Rule Learning* (ARL) [23] to answer the first two questions. RefactoringMiner is currently considered the state-of-the-art refactoring detection tool (precision of 97.96% and recall of 87.2% [78], whereas ARL can discover non-obvious relationships between variables in large datasets [12]. We used RefactoringMiner to detect refactorings in a sample of 1,845 merged PRs. Then, we performed ARL on two groups (refactoring-inducing and non-refactoring-inducing PRs), and formulated eight (8) hypotheses on differences between refactoring-inducing and non-refactoring-inducing PRs by manually exploring 562 association rules discovered by ARL. We found that refactoring-inducing PRs significantly differ from non-refactoring-inducing ones in terms of number of subsequent commits, code churn, number of file changes, number of review comments, length of discussion, and time to merge; however, we found no statistical evidence that the number of reviewers is related to refactoring-inducement.

In order to address the third research question, we carried out a manual investigation of 2,096 review comments cross-referenced to 1,891 detected refactorings from 228 refactoring-inducing PRs – a stratified sample from our original sample (by considering a confidence level of 95% and a margin of error of 5%). We found 133 refactoring-inducing PRs (58.3%) in which at least one refactoring edit was induced by review comments.

Contributions:

(1) To the best of our knowledge, this is the first study investigating aspects related to refactoring and code review in the context of refactoring-inducing PRs (Def. 1.1).
(2) We investigate PRs merged by *merge pull request* and *squash and merge* options. We tried to avoid either PRs merged by *rebase and merge* or merged PRs that suffered rebasing, intending to minimize threats to validity (Section 4.1). To deal with squashed commits, we implemented a script that recovers them (*git squash* converts all commits in a PR into a single commit).

(3) We performed a manual analysis of refactoring-inducement, by exploring more than 2,000 review comments.
(4) We made available a complete reproduction kit [10] including the mined dataset and implemented scripts to enable replications and future research.

2 BACKGROUND

2.1 Refactoring and Modern Code Review

As software evolves to meet new requirements, its code becomes more complex. Throughout this process, design and quality deserve attention [44]. For that, code restructurings, coined as refactorings by Opdyke and Johnson [57], are performed to improve the design quality of object-oriented software, while preserving its external behavior, and they should be performed in a structured manner [33, 56]. Developers can recover those restructurings through refactoring detection tools – which automatically identify refactoring types applied to the code, for assisting tasks such as studies on code evolution [60] and MCR [14, 35]. MCR consists of a lightweight code review (in opposition to the formal code inspections specified by Fagan [32]), tool-assisted, asynchronous, and driven by reviewing code changes, submitted by a developer (author), and manually examined by one or more other developers (reviewers) [18].

2.2 Git-Based Development and Pull Requests

Git-based collaborative development as implemented in GitHub [8] has presented a fast growth in the number of developers (more than 56 million) [4]. Each Git repository maintains a full history of changes [29] structured as a linked-list of commits, in turn, organized into multiple lines of development (branches). A PR is a commonly used way for submitting contributions to collaboration-based projects [9]. After forking a Git branch, a developer can implement changes, and open a PR to submit them for reviewing in line with the MCR process. Next, reviewers can submit comments based on a diff output that highlights the changes, whereas the author and other contributors can answer the reviewers' comments. After the reviewing, there are three options of merging:

- *Merge pull request* merges the PR commits into a *merge commit* and adds them into the main branch, chronologically ordered, as depicted in Figure 2. Note that the arrows indicate a commit's parent, and the *before* and *after* markers indicate the commits searchable in the PR, respectively, before and after merging;

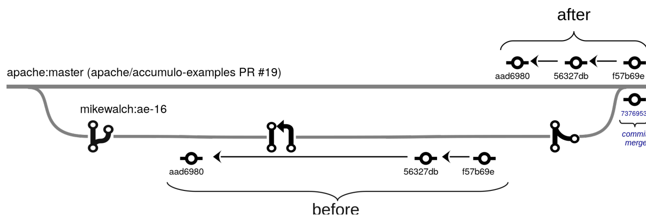


Figure 2: Illustrating Merge Pull Request Option (Apache accumulo-examples PR #19)

- *Squash and merge* squashes the PR commits into a single commit and merges it into the main branch (Figure 3); and
- *Rebase and merge* re-writes all commits from one branch onto another, by updating their SHA, in a manner that unwanted history can be discarded, as illustrated in Figure 4. In this case, commits *0be3d3f* and *66f02d3* received review comments, but they are not accessible via PR. Hence, it is mandatory to recover the original commits when investigating reviewing-related aspects. Nonetheless, such a recovery is not trivial [42].

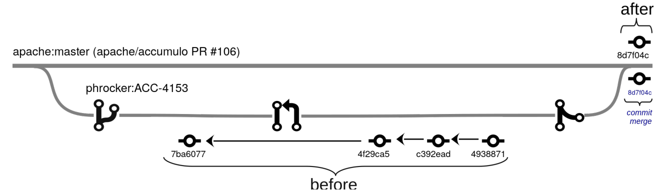


Figure 3: Illustrating Squash and Merge Option (Apache accumulo PR #106)

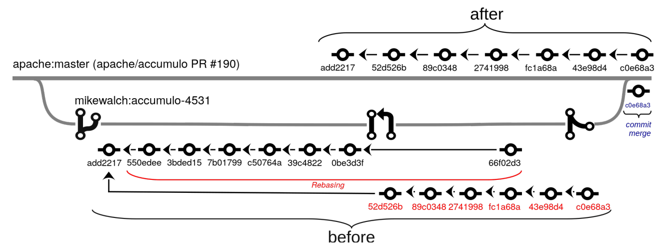


Figure 4: Illustrating Rebase and Merge Option (Apache accumulo PR #190)

2.3 Association Rule Learning

ARL discovers rules that denote non-obvious relationships between variables in large datasets, e.g., refactoring-inducing PRs with a high number of added lines tend to have a high number of reviewers. Formally, let $I = \{i_1, i_2, ..., i_n\}$, a set of n binary attributes (*items*) and $D = \{t_1, t_2, ..., t_m\}$, a set of m transactions (*dataset*), in which each transaction in D consists of items in I. Thus, an *Association Rule* (AR) $\{X\} \rightarrow \{Y\}$ indicates the co-occurrence of the tuples $\{X\}$ (*antecedent*) and $\{Y\}$ (*consequent*), where $\{X\}, \{Y\} \subseteq I$, $\{X\} \cap \{Y\} = \emptyset$ [12]. *Support* indicates the number of transactions in D that supports an AR, so expressing its statistical significance.

Interestingness measures can determine the strength of an AR. *Confidence* means how likely $\{X\}$ and $\{Y\}$ will occur together. *Lift* reveals how X and Y are related to one another (0 denotes no association, < 1 indicates a negative co-occurrence of the antecedent and consequent, and > 1 express that the two occurrences are dependent on one another and the ARs are useful) [36]. *Conviction* is a measure of implication, ranging in the interval $[0, \infty]$. Conviction 1 denotes that antecedent and consequent are unrelated, while ∞ expresses logical implications, where confidence is 1 [26].

ARL usually follows this workflow: feature selection, feature engineering (applying any encoding technique, such as *one-hot encoding* using a group of bits to represent mutually exclusive features [80]), algorithm choice and execution, and result interpretation (assisted by interestingness measures) [79].

3 MOTIVATING EXAMPLE

This study has evolved from results of preliminary investigations on refactorings and code reviews to get a better understanding of

the topic and plan the research design. As a motivating example, we describe a case history, in which we explored the refactoring-inducement and code review aspects. We randomly selected 24 PRs from Apache's drill repository. Then, we ran RefactoringMiner and obtained 11 (45.8%) refactoring-inducing PRs.

We compared refactoring-inducing and non-refactoring-inducing PRs concerning *code churn* (number of changed lines), and discussion length (i.e., review and non-review comments). As a result, we identified that the refactoring-inducing PRs presented a higher code churn and discussion length than non-refactoring-inducing PRs. Note that we took into account one measure of each context under investigation: changes (code churn), code review (length of discussion), besides the number of refactoring edits.

We manually analysed the refactoring-inducing PRs, by contrasting the descriptions of the detected refactorings by RefactoringMiner against review comments. Our strategy of analysis consisted of reading comments and searching for keywords (e.g., "refac", "mov", "extract", and "renam"). We observed refactorings directly induced by review comments in four refactoring-inducing PRs. To exemplify, in PR #1762[1], the review comment "*Lot of code here and in DefaultMemoryAllocationUtilities are duplicate. May be create a separate MemoryAllocationUtilities to keep the common code...*" motivated one *Extract Superclass* and four *Pull Up Method* refactorings.

In a nutshell, those results provided insights on the pertinence of (i) exploring technical aspects of changes, code review, and refactorings in the PR level, since we perceived differences between refactoring-inducing and non-refactoring-inducing PRs in terms of code churn and length of discussion; (ii) considering refactorings as part of contributions to the code improvement during code review, and (iii) investigating quantitatively and qualitatively technical aspects in light of the refactoring-inducing PR definition.

4 STUDY DESIGN

The main goal of this study is to investigate code reviewing-related data to characterize refactoring-inducing PRs in Apache's repositories hosted in GitHub, from the reviewers' perspective. Thus, we formulated these **research questions**:

- RQ_1: How common are refactoring-inducing PRs? We firstly explored the presence of PRs that met our refactoring-inducing PR definition (Def. 1.1).
- RQ_2: How do refactoring-inducing PRs compare to non-refactoring-inducing ones? We quantitatively investigated code reviewing-related aspects aiming to find out similarities/differences in PRs based on the refactorings performed.
- RQ_3: Is refactoring induced by code reviews? We qualitatively scrutinized a stratified sample of refactoring-inducing PRs to validate the occurrence of refactoring edits induced by code reviewing, by manually examining review comments and discussions.

Accordingly, supported by guidelines [70], we designed an empirical study that comprises five steps, as shown in Figure 5 and described in the next subsections. Also, we made publicly available a reproduction kit containing the mined datasets and developed scripts for replicating the results for our research questions [10].

[1]Apache drill PR #1762, available in https://git.io/JczHh.

4.1 Mining Merged Pull Requests

We mined merged PRs from Apache's repositories at GitHub. We focused on merged PRs because they reveal actions that were in fact finalized, therefore, we can get a more in-depth understanding of refactoring-inducement. We chose GitHub due to its popularity [4] and to the mining resources available through extensive APIs – GitHub REST API v3 [7] and GitHub GraphQL API v4 [6].

The *Apache Software Foundation* (ASF) manages more than 350 open-source projects, with more than 8,000 contributors from all over the world; all of its projects migrated to GitHub in February 2019 [2]. Given Apache's popularity and relevance of contributions in the open-source software development context, we selected it for mining PRs [5]. The refactoring mining tool we selected (Section 4.2) only supports projects developed in the Java, so we considered Java projects (almost 57% of Apache's code is developed in Java).

In August 2019, we searched on Apache's non-archived Java repositories in GitHub (to take into account only actively maintained repositories), resulting in 65,006 merged PRs, detected in 467 out of 956 repositories; we then implemented a script to mine their merged PRs. We obtained two datasets: *pull requests dataset* consists of 48,338 merged PRs (*merge PR option*) from 453 distinct repositories while *commits dataset* contains 53,915 recovered commits from 16,668 merged PRs (*squash and merge* or *rebase and merge* options) from 255 repositories.

Then, we recovered the commit history of squashed and merged PRs before any exploration of its original commits, assisted by the *HeadRefForcePushedEvent* object accessible via GitHub GraphQL API [6]. To clarify, consider the Apache's PR 1807 (Figure 6) that, originally, had 12 commits ($c_1 - c_{12}$) that were squashed into single commit ($c_{afterCommit}$) after a *force–pushed* event. Consequently, only one commit may be gathered from the PR ($c_{afterCommit}$).

Our recovery strategy follows two steps: (1) we recover the commits $c_{afterCommit}$ and $c_{beforeCommit}$ through *HeadRefForcePushedEvent* object; and (2) we rebuild the original commits' history through tracking the commits from $c_{beforeCommit}$, which has the same value of c_{12}, until reaching the same SHA of the $c_{afterCommit}$'s parent, by using the *compare* operation, as available in GitHub REST API v3 [7]. We executed the strategy's Step 1 for gathering the after and before commits from 65,006 pull requests, obtaining 53,915 commits after running the strategy's Step 2.

We discarded PRs merged by *rebase and merge* option since, in rebasing, some commits within the PR may be due to external changes (outside the scope of the code review sequence), conveying a threat to the validity, as argued in [59]. Accordingly, we considered the number of *HeadRefForcePushedEvent* events and PR commits to identify PRs merged by *squash and merge*. In specific, PRs merged by *merge pull request* and *squash and merge* present zero and one *HeadRefForcePushedEvent* event, respectively (squashed and merged PRs keep one PR commit). Moreover, we dropped all PRs containing at least one subsequent commit with two parents, because such commits may represent external changes rebased onto a branch, as depicted in Figure 7. Note that, once commit *ee88dea* has two parents, it integrates external changes, which were not reviewed in PR reviewing time.

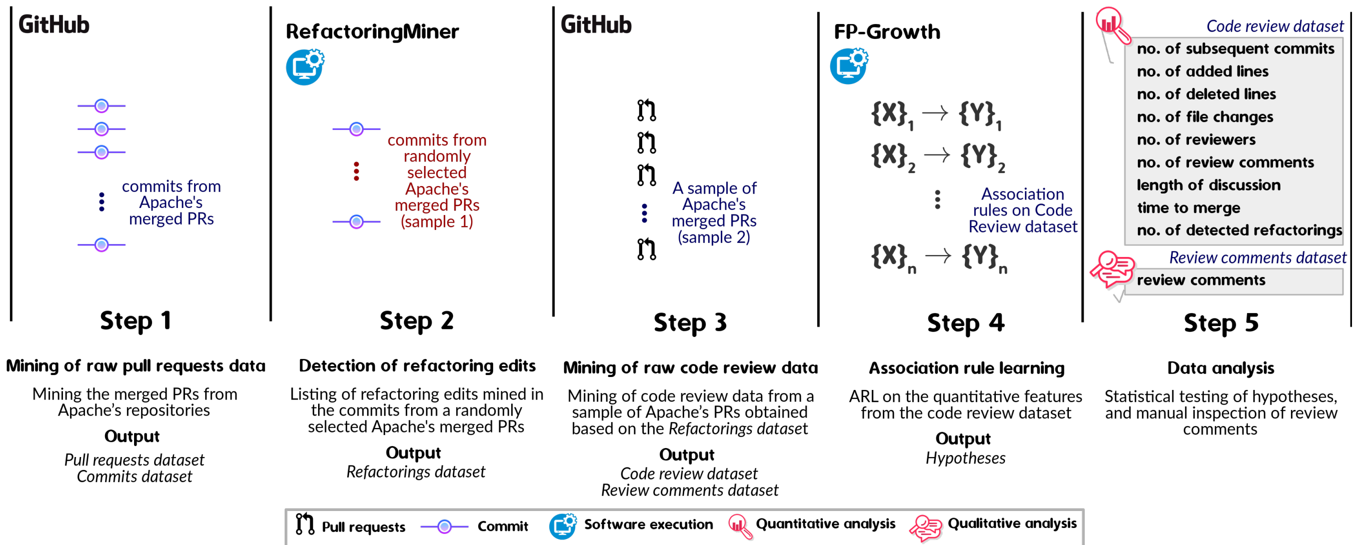


Figure 5: Overview of our Investigation.

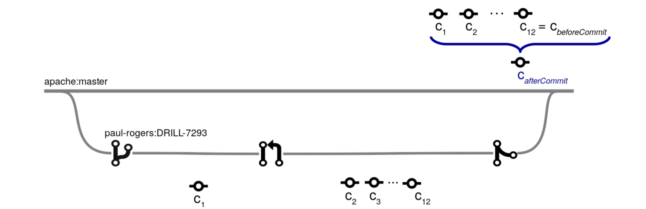


Figure 6: An Overview of Apache Drill PR #1807, Illustrating Squashed Commits ($c_1 - c_{12}$).

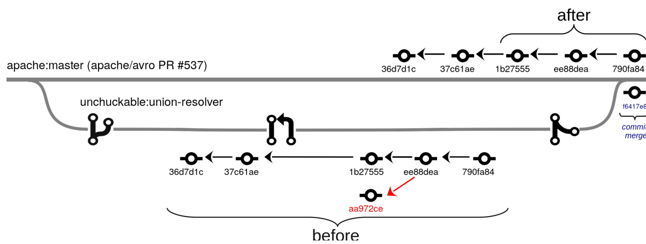


Figure 7: Illustrating a Pull Request's Commit Presenting Two Parents (Apache avro PR #537)

4.2 Refactoring Detection

RefactoringMiner detects refactorings in Java projects, presenting better results when compared to its competitors (precision of 99.6% and recall of 94%) [77, 78]. We considered version 2.0, which supports over 40 different refactoring types, including low-level refactorings, such as variable renames and extractions, allowing us to work with a more comprehensive list of refactoring edits. For these reasons, we selected it for refactoring detection (Step 2). In essence, it identifies the refactorings performed in a commit in relation to its parent commit, displaying a description of the applied refactorings (type and associated targets, e.g., the methods and classes involved in an *Extract and Move Method* refactoring). In this step, we considered only merged PRs containing two or more commits (*sample 1*, Figure 5) intending to conform with our refactoring-inducing PR definition. After three weeks of RefactoringMiner running, we obtained a random sample of 225,127 detected refactorings in 8,761 merged PRs (13.5% of the total number of Apache's merged PRs) from 209 distinct repositories, embracing 68,209 commits. The source of randomness lies in the order in which the repositories were processed.

At that point, we checked the *commits' authored date* against the *PRs' opening date* in order to identify initial and subsequent commits for the sample's PRs. Therefore, the number of refactorings of a PR takes into account only subsequent commits.

4.3 Mining Code Review Data

Empirical studies have investigated code review efficiency and effectiveness to understand the practice, elaborate recommendations, and develop improvements. Together, these studies share a set of useful code review aspects for further investigation, such as change description [25, 75], code churn [76], length of discussion [45, 54, 66, 76], number of changed files [25, 45], number of commits [54, 67], number of people in the discussion [45], number of resubmissions [45, 66], number of review comments, [21, 54, 66], number of reviewers [66, 72], size of change [20, 45, 66], and time to merge [37, 41]. Therefore, the mining of raw code review data (Step 3) consisted of collecting the code reviewing-related attributes listed in Table 1, considering 8,761 PRs from Step 2 (*sample 2*, Figure 5). Attributes number, title, labels, and repository's name are useful to uniquely identify a PR. We clarify that we do not count the distinct files changed (i.e., the set of the changed files), but the number of times the files changed (i.e., the list of file changes) over subsequent PR commits. Hence, the number of added lines and deleted lines denote the number of lines modified across file changes.

For mining, we imposed one *precondition*: only merged PRs, comprising at least one review comment, should be mined aiming to explore refactoring-inducement and to collect review comments for further investigation. Thus, the mining generated two datasets, *code review dataset* and *review comments dataset*, refined according to the

Table 1: Selected Pull Request Attributes for Mining

Attribute	*Description*
number	Numerical identifier of a PR
title	Title of a PR
repository	Repository's name of a PR
labels	Labels associated with a PR
commits	No. of subsequent commits in a PR
additions	No. of added lines in a PR
deletions	No. of deleted lines in a PR
file changes	No. of file changes in a PR
creation date	Date and time of a PR creation
merge date	Date and time of a PR merge
review comments	No. of review comments in a PR
non-review comments	No. of non-review comments in a PR

following procedures: dropping merged PRs with inconsistencies, such as zero file changes and zero reviewers; checking for duplicates; and mining from non-mirrored repositories. As a result, our final sample consists of code review data from 1,845 merged PRs (2.8% of the total number of Apache's merged PRs from Step 1 and 21.1% of the number of sample's PRs obtained from Step 2), encompassing 4,702 subsequent commits, 6,556 detected refactorings, and 12,547 review comments, mined from 84 distinct Apache's repositories.

4.4 Association Rule Learning

Aiming to explore what differentiates refactoring-inducing PRs from non-refactoring-inducing ones, we executed ARL (Step 4). Such strategy assists exploratory analysis by identifying natural structures derived from the relationships between the characteristics of data [28]. Accordingly, by considering ARL on refactoring-inducing and non-refactoring-inducing PRs, we can identify ARs that likely support us in the formulation of more accurate hypotheses concerning differences/similarities between those two groups. One may argue that clustering is a better alternative than ARL to find groups of PRs with distinct characteristics. Nonetheless, we experimentally performed clustering in our sample of PRs, after conducting a rigorous selection of clustering algorithm and input parameters[2], but we found a great noise ratio (76.3%).

4.4.1 Selection of features. We selected all features that can be represented as a number regarding changes, code review, and refactorings, from the code review dataset (Step 3). We considered a three-context perspective (changes, code review, and refactorings) because they together might potentially support the identification of differences between refactoring-inducing and non-refactoring-inducing PRs. These are the selected features: number of subsequent commits, number of file changes, number of added lines, number of deleted lines, number of reviewers, number of review comments, length of discussion, time to merge, and number of detected refactorings. Note that the length of discussion and time to merge are derived from *review comments* + *non-review comments*, and *merge date* − *creation date* (in number of days), respectively.

One may argue that other features could also be considered; however, (i) the PR title is written using natural language, so it is subject to ambiguities; (ii) PR labels are not mandatory, only 349 PRs from our sample have labels; (iii) date and time of creation/merge are specific values, so we used the difference between them (time to merge) for exploration; and (iv) the number of non-review comments of a PR is part of its length of discussion.

[2] *We used the Ordering Points To Identify the Clustering Structure* (OPTICS) algorithm [15] and *Euclidean distance* [16] as similarity metric.

Table 2: One-Hot Encoding for Binning of Features

Category	*Range*
None	0
Low	$0 < quantile \leq 0.25$
Medium	$0.25 < quantile \leq 0.50$
High	$0.50 < quantile \leq 0.75$
Very high	$0.75 < quantile \leq 1.0$

Table 3: ARL Output by Experimenting Minimum Support from 0.01 to 0.1 by Steps of 0.01, and Confidence of 0.5

Support ≥	*Number of association rules*
0.01	52,944
0.02	19,239
0.03	10,354
0.04	5,567
0.05	3,572
0.06	2,264
0.07	1,640
0.08	1,004
0.09	712
0.10	562

4.4.2 Feature engineering. We applied one-hot encoding based on the quartiles of the features, resulting in the binning presented in Table 2. We chose such technique due to its simplicity and linear time and space complexities [80]. We did not discard the outliers because, in the context of this study, they do not represent experimental errors; thus, they can potentially indicate circumstances for further examination. Consequently, the *very high* category (fourth quartile) includes the outliers.

4.4.3 Selection and execution of an algorithm. We selected the FP-Growth algorithm due to its performance [39]. Then, we developed a script for the ARL by using the FP-growth implementation available in the *mlxtend.frequent_patterns* module [64]. We set the minimum support threshold to 0.1 to avoid discarding likely ARs for further analysis [30]. Aiming to get meaningful ARs, we considered minimum thresholds for confidence ≥ 0.5, lift > 1, and conviction > 1. We performed a prior experiment concerning values of minimum support and minimum confidence by taking the thresholds considered in [12] as a reference (support of 0.01, confidence of 0.5). We ran FP-growth considering support values ranging from 0.01 to 0.1 by steps of 0.01, and confidence 0.5 (Table 3). In all these settings, we found ARs that cover all input features. Since support is a statistical significance measure, we consider the last setting (minimum support of 0.1, confidence of 0.5) for purposes of FP-growth execution. A lift threshold > 1 reveals useful ARs [22], while a conviction threshold > 1 denotes ARs with logical implications [26].

4.4.4 Interpretation of results. We considered the feature levels (*none*, *low*, *medium*, *high*, and *very high*), instead of absolute values,

as items for composing ARs aiming to identify relative associations among two groups for investigation, e.g., {*high number of added lines*} → {*high number of reviewers*}. The ARs work as basis for the formulation of hypotheses regarding the characterization of our sample's PRs. In this sense, we carried out the following procedure:

(1) manual examination of the ARs to recognize potential differences/similarities that support the formulation of hypotheses;
(2) analysis of the pairwise ARs, ARs containing the *number of refactorings* as an item, and ARs whose conviction is infinite to assist the rationale for the formulation of hypotheses; and
(3) formulation of hypotheses to quantitatively investigate the differences between refactoring-inducing and non-refactoring-inducing PRs.

4.5 Data Analysis

4.5.1 Quantitative data analysis. We analyzed the output of Step 3 by exploring the detected refactorings by PR to answer RQ_1. The number of refactorings was computed by considering the edits detected as in the PR *subsequent* commit(s). As a complement, we computed a 95% confidence interval for the percentual (proportion) of refactoring-inducing PRs in Apache's merged PRs, by performing *bootstrap resampling* [31]. We applied statistical testing of hypotheses intending to answer RQ_2. That analysis encompassed the testing of eight hypotheses formulated from the analysis of the ARL output (Step 4), driven by a comparison between refactoring-inducing and non-refactoring-inducing PRs. We executed each hypothesis testing in line with this workflow, guided by [27]:

(1) Definition of null and alternative hypotheses.
(2) Performing of statistical test. We considered a significance level of 5%, and a substantive significance (effect size) for denoting the magnitude of the differences between refactoring-inducing and non-refactoring-inducing PRs at the population level. First, we checked the assumptions for parametric statistical tests (steps *a* and *b*), since the independence assumption is already met (i.e., a PR is either a refactoring-inducing or not). For exploring the difference between refactoring-inducing and non-refactoring-inducing PRs, we computed a 95% confidence interval by bootstrapping resample according to the output from steps *a* and *b*, in mean or median (step *c*). Then, we conducted a proper statistical test and calculated the effect size (step *d*).
    (a) checking for data normality by using the *Shapiro-Wilk* test;
    (b) checking for homogeneity of variances via *Levene's* test;
    (c) computation of confidence interval for the difference in mean or median aligned to output from steps *a* and *b*;
    (d) performing of either parametric independent *t-test* and *Cohen's d*, or non-parametric *Mann Whitney U* test and *Common-Language Effect Size* (CLES) in line with the output from steps *a* and *b*. CLES is the probability, at the population level, that a randomly selected observation from a sample will be higher/greater than a randomly selected observation from another sample [53].
(3) Deciding if the null hypothesis is supported or refused.

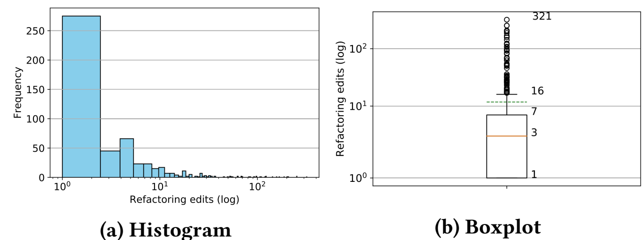
4.5.2 Qualitative data analysis. In order to answer RQ_3, three developers (intending to mitigate researcher bias) manually examined review discussions and validated the detected refactorings from a subset of refactoring-inducing PRs of our sample. We adopted a stratified random sampling to select refactoring-inducing PRs for an in-depth investigation of their review comments and discussion while cross-referencing their detected refactoring edits. Moreover, we validated these refactorings by checking for false positives. As a whole, the qualitative analysis lasted 30 days. We chose that sampling strategy because it provides a means to sample non-overlapping subgroups based on specific characteristics [52], (e.g. number of refactorings), where each subgroup (*stratum*) can be sampled using another sampling method – a setting that quite fits to further investigation of categories of refactoring-inducing PRs containing a *low*, *medium*, *high*, and *very high* number of refactorings (Table 2). To define the sample size, we considered a confidence level of 95% and a margin of error of 5%, so obtaining 228, thus considering 57 refactoring-inducing PRs randomly selected from each category. We split the samples into four categories based on the numbers of refactorings in order to check if there is a difference in the effect of code review refactoring requests/inducement between PRs with massive refactoring efforts versus PRs with small/focused refactoring efforts.

In the analysis, firstly, we conducted a calibration in which one of the analysts followed up ten analyses performed by the others. Next, each analyst apart examined 40.3%, 38.2%, and 21.5% of the data. In such subjective decision-making, we considered the refactoring-inducement in settings where review comments either explicitly suggested refactoring edits (e.g., "How about renaming to ...?"[3]) or left any actionable recommendation that induced refactoring (e.g., "avoid multiple booleans" induced a *Merge Parameter* instance[4]).

5 RESULTS AND DISCUSSION

5.1 How Common are Refactoring-Inducing Pull Requests?

We found 557 refactoring-inducing PRs (30.2% of our sample's PRs), equaling 12,547 detected refactoring edits. As shown in Figure 8a, the histogram of refactoring edits is positively skewed, presenting outliers. Thus, a low number of refactoring edits is quite frequent. The number of refactorings by PR is 11.8 on average (SD = 32.3) and 3 as median (IQR = 6), according to Figure 8b.



(a) Histogram **(b) Boxplot**

Figure 8: Refactorings in the Refactoring-Inducing PRs

Using bootstrapping resampling and a 95% confidence level, we obtained a confidence interval ranging from 28.1% to 32.3% for the proportion of refactoring-inducing PRs in Apache's merged PRs. These results reveal significant refactoring activity induced in PRs. This is a motivating result, while outliers' presence can indicate scenarios scientifically relevant for further exploration.

[3] Apache samza PR #1051, available in https://git.io/J3z9H.
[4] Apache fluo PR #1032, available in https://git.io/J3mxZ.

Finding 1: We found 30.2% of refactoring-inducing PRs, which percentage (proportion) in Apache's merged PRs is in [28.1%, 32.3%], for a 95% confidence level.

5.2 How Do Refactoring-Inducing Pull Requests Compare to non-Refactoring-Inducing Ones?

From ARL, we obtained 562 ARs (146 from refactoring-inducing PRs and 416 from non-refactoring-inducing PRs). Then, we manually inspected them, by searching for pairwise ARs (AR_1–AR_7), ARs whose conviction is infinite (AR_5, AR_6), and the remaining ARs (AR_2, AR_3, AR_4). Accordingly, we selected four ARs (AR_1–AR_4) obtained from refactoring-inducing PRs and three ARs (AR_5–AR_7) from non-refactoring-inducing PRs, all catalogued in Table 4, in decreasing order of conviction. Since we did not identify the same pairs of ARs in both groups, we needed to consider a distinct number of ARs (hence, itemsets) for the comparison purpose when addressing all features. Afterwards, we carried out an analysis of those ARs. We formulated eight hypotheses on the differences/similarities between refactoring-inducing and non-refactoring-inducing PRs, discussed as follows. Table 5 shows the *average*, *Standard Deviation* (SD), *median*, and *Interquartile Range* (IQR) of the examined features from refactoring-inducing and non-refactoring-inducing PRs.

***H₁*. Refactoring-inducing PRs are more likely to have more added lines than non-refactoring-inducing PRs** (AR_2/AR_3,AR_5).

***H₂*. Refactoring-inducing PRs are more likely to have more deleted lines than non-refactoring-inducing PRs** (AR_2/AR_3,AR_5).

Finding 2: Refactoring-inducing PRs comprise significantly more code churn than non-refactoring-inducing ones, since refactoring-inducing PRs are significantly more likely to have a higher number of added lines (U = $0.58 \times e^{+06}$, $p < .05$), CLES = 81.2% and deleted lines (U = $0.57 \times e^{+06}$, $p < .05$), CLES = 80.5% than non-refactoring-inducing PRs.

This is an expected result in light of the findings from Hegedüs et al., since refactored code has significantly higher size-related metrics [40]. We speculate that reviewing larger code churn may potentially promote refactorings. This understanding is supported by Rigby et al., who observed that the code churn's magnitude influences code reviewing [67, 68], and Beller et al. who discovered that the larger the churn, the more changes could follow [21].

***H₃*. Refactoring-inducing PRs are more likely to have more file changes than non-refactoring-inducing PRs** (AR_2/AR_3,AR_5).

Finding 3: Refactoring-inducing PRs encompass significantly more file changes than non-refactoring-inducing ones (U = $0.56 \times e^{+06}$, $p < .05$), CLES = 79.1%.

We conjecture that reviewing code arranged across files may motivate refactorings, an argument supported by Beller et al. regarding more file changes comprise more changes during code review [21]. By observing change-related aspects (churn and file changes), our findings confirm previous conclusions on the influence of the amount and magnitude of changes on code review [20, 45, 67, 68]. When analyzing the changes and refactorings, our findings reinforce prior conclusions on refactored code significantly present higher size-related metrics (e.g., number of code lines and file changes) [40], and larger changes promote refactorings [58].

***H₄*. Refactoring-inducing PRs are more likely to have more subsequent commits than non-refactoring-inducing PRs** (AR_2/AR_3, AR_5).

Finding 4: Refactoring-inducing PRs comprise significantly more subsequent commits than non-refactoring-inducing PRs (U = $0.51 \times e^{+06}$, $p < .05$), CLES = 70.6%.

Based on our previous findings on the magnitude of code churn and file changes, that result is expected and aligned to Beller et al. concerning the impacts of larger code churn and wide-spread changes across files on consequent changes [21]. Accordingly, we speculate that reviewing refactoring-inducing PRs might require more subsequent changes, in turn, denoted by more subsequent commits in comparison with non-refactoring-inducing PRs.

***H₅*. Refactoring-inducing PRs are more likely to have more review comments than non-refactoring-inducing PRs** (AR_1, AR_7).

Finding 5: Refactoring-inducing PRs embrace significantly more review comments than non-refactoring-inducing PRs (U = $0.47 \times e^{+06}$, $p < .05$), CLES = 65.1%.

Beller et al. found that the most changes during code review are driven by review comments [21], and Pantiuchina et al. discovered that almost 35% of refactoring edits are motivated by discussion among developers in OSS projects at GitHub [61]. Thus, we conjecture that, besides change-related aspects, GitHub's PR model can constitute a peculiar structure for code review, in which review comments influence the occurrence of refactorings, therefore explaining our result. This argument originates from the fact that a pull-based collaboration workflow provides reviewing resources [9] (e.g., a proper code reviewing UI) for developers to improve/fix the code while having access to the history of commits and discussion. Our finding also provides insight for examination of review comments to get an in-depth understanding of refactoring-inducement.

***H₆*. Refactoring-inducing PRs are more likely to present a lengthier discussion than non-refactoring-inducing PRs** (AR_1, AR_7).

Finding 6: Refactoring-inducing PRs enclose significantly more discussion than non-refactoring-inducing PRs (U = $0.46 \times e^{+06}$, $p < .05$), CLES = 64.7%.

A more in-depth analysis could tell how profound these lengthier discussions are, although a higher number of comments might represent developers concerned with the code, willing then to extend their collaboration to the suggestion of refactorings. Previous findings may support those claims; Lee and Cole, when studying the Linux kernel development, acknowledged that the amount of discussion is a quality indicator [48]. Also, empirical evidence reports on the impact of the number of comments on changes [21, 61].

Table 4: Association Rules Selected by Manual Inspection (AR_1–AR_4 for Refactoring-Inducing PRs, AR_5–AR_7 for non-Refactoring-Inducing PRs)

Id	*Association rule*	*Supp*	*Conf*	*Lift*	*Conv*
AR_1	{very high length of discussion, very no. of reviewers} → {very high no. of review comments}	0.13	0.85	3.08	4.89
AR_2	{very high no. of added lines, very high no. of subsequent commits} → {very high no. of file changes}	0.11	0.83	3.23	4.51
AR_3	{very high no. of deleted lines, very high no. of subsequent commits} → {very high no. of file changes}	0.10	0.81	3.12	3.81
AR_4	{medium time to merge} → {very high no. of reviewers}	0.16	0.51	1.06	1.06
AR_5	{high no. of subsequent commits, low no. of added lines, low no. of deleted lines} → {medium no. of file changes}	0.12	1.00	2.63	∞
AR_6	{medium no. of file changes, very high no. of reviewers, medium time to merge} → {high no. of subsequent commits}	0.13	1.00	1.83	∞
AR_7	{very high no. of reviewers, medium length of discussion} → {medium no. of review comments}	0.13	0.61	1.71	1.63

Table 5: Statistics of the Pull Requests Attributes

Pull Request Attribute	***Refactoring-Inducing Pull Requests***				***non-Refactoring-Inducing Pull Requests***			
	Average	SD	Median	IQR	Average	SD	Median	IQR
Number of added lines	945.9	4,744.3	72	250	57.5	517.8	8	28
Number of deleted lines	377.4	1,859.7	41	139	41.2	303.8	6	16.2
Number of file changes	32.1	119.7	7	15	6.1	60.2	2	3
Number of subsequent commits	3.7	3.4	3	2	2.1	1.9	1	1
Number of review comments	9.8	11.1	6	9	5.5	8.2	3.5	4
Length of discussion	15.2	13.8	11	14	10.1	12.1	7	8
Number of reviewers	2.3	0.9	2	1	2.1	0.9	2	0
Time to merge (days)	14.3	45.6	5	11	9.3	33.1	2	7

***H_7*. Refactoring-inducing and non-refactoring-inducing PRs are equally likely to have a higher number of reviewers** (AR_1, AR_7).

Finding 7: We found no statistical evidence that the number of reviewers is related to refactoring-inducement (U = $0.40 \times e^{+06}$, $p < .05$), CLES = 55.9%.

Refactoring-inducing and non-refactoring-inducing PRs present two reviewers as median − the same result found by Rigby et al. [65] in the OSS scenario. There are outliers that, in turn, could be justified by other technical factors, such as complexity of changes, as argued in [66]. However, our study does not address that scope.

***H_8*. Refactoring-inducing PRs are more likely to take a longer time to merge than non-refactoring-inducing PRs** (AR_4, AR_6).

Finding 8: Refactoring-inducing PRs take significantly more time to merge than non-refactoring-inducing PRs (U = $0.42 \times e^{+06}$, $p < .05$), CLES = 59.3%.

We realize the influence of refactorings on time to merge, concluding that time for reviewing and performing refactoring edits both impact the time to merge. In special, this conclusion is aligned to Szoke et al., who observed a correlation between implementing refactorings and time [74], and from Gousios et al., who found that review comments and discussion affect time to merge a PR [37].

5.3 Is Refactoring Induced by Code Reviews?

To study this research question, we sampled 228 refactoring-inducing PRs, 57 PRs from each of the *Low*, *Medium*, *High*, and *Very High* categories encompassing one, two to three, four to seven, and eight to 321 refactoring edits, respectively. By examining 2,096 review comments and 1,207 discussion comments in the sampled PRs, we found 133 (58.3%) in which at least one refactoring edit was induced by review comments. Such PRs comprise 815 subsequent commits, and 1,891 detected refactorings, 545 of which were induced by review comments. Finally, we found that *Rename* (35.8%) (being *readability* a common motivation cited by reviewers) and *Change Type* (30.3%) operations are the most induced by review in our stratified sample.

Finding 9: In a stratified sample of 228 refactoring-inducing PRs, 133 ones (58.3%) presented at least one refactoring edit induced by code review.

5.4 Implications

Researchers: All our findings, except for Finding 7, indicate that refactoring-inducing and non-refactoring-inducing PRs have different characteristics. Therefore, we recommend that future experiment designs on MCR with PRs to *make a distinction between refactoring-inducing and non-refactoring-inducing PRs, or consider their different characteristics when sampling PRs*. Researchers can also use our mined data, developed tools, and research methods to investigate code reviewing in pull-based development.

Practitioners: Our findings indicate that there is no statistical difference in the number of reviewers between refactoring-inducing and non-refactoring-inducing PRs (Finding 7). But, all other findings show that refactoring-inducing PRs are associated with more churn (Finding 2), more file changes (Finding 3), more subsequent commits (Finding 4), more review comments (Finding 5), lengthier discussions (Finding 6), and more time to merge (Finding 8) than non-refactoring-inducing PRs. Thus, *we suggest to PR managers to invite more reviewers when a PR becomes refactoring-inducing*, to

share the expected increase in review workload, and, perhaps more importantly, to share the knowledge of design changes caused by subsequent refactorings to more team members.

Tool builders: In connection to our implication for practitioners, tool builders can *develop bots* [47, 73] *that recommend reviewers based on some criteria* [55] *when a PR becomes refactoring-inducing, to assist the PR managers in inviting additional reviewers.* Our findings indicate that refactoring-inducing PRs have higher complexity in churn (Finding 2) and file changes (Finding 3). *Therefore, it is necessary to help the developers distinguish refactoring edits from non-refactoring edits directly in the GitHub or Gerrit review board, where the reviews are actually taking place.* In the past, researchers implemented refactoring-awareness in the code diff mechanism of IDEs [13, 34, 35]. Even though not directly related to our results, we believe that adding refactoring-awareness directly in the GitHub or Gerrit review board – such as the refactoring-aware commit review Chrome browser extension [51] – would allow reviewers to trace the refactorings performed throughout the commits of a PR, provide prompt feedback, and concentrate efforts on other aspects of the changes, such as collateral effects of refactorings and proposing specific tests. This recommendation is in agreement with Gousios et al. [38], who emphasized the need for untangling code changes and supporting change impact analysis directly in the PR interface.

6 THREATS TO VALIDITY

We elaborated our study design after conducting two case studies to better understand GitHub's PRs and procedures of data mining and refactoring detection. We carefully defined workflows for our research design procedures to explain all decisions taken, and we systematically structured all procedures aiming at replicability. We performed a rigorous selection of the ARL algorithm and input parameters. To mitigate researcher bias, our qualitative analysis was performed by three analysts. Despite our efforts to perform an initial calibration, there may be limitations concerning conclusions, since we carried out apart analyses.

Nevertheless the establishment of a chain of evidence for the data interpretation and description of taken decisions in the study design, we did not validate the detected refactorings before data analysis, so expressing a potential threat to construct validity (RQ_1 and RQ_2). To overcome this issue, we selected RefactoringMiner, a state-of-the-art refactoring detection tool [78]. When addressing RQ_3, we validated all detected refactorings in our stratified sample.

Aiming to mitigate the risk related to rebasing constraints in our sample, we excluded the PRs merged with the *rebase and merge* option and the PRs including intermediate *merge commits*. Even so, there are still threats due to other non-previously identified manners to search for rebasing operations.

Furthermore, as already admitted in the refactoring-inducing PR definition, we cannot claim that all refactoring edits were caused by reviewing. To deal with such limitation, we carried out a qualitative analysis of review comments from 228 randomly selected refactoring-inducing PRs, considering a sample size meeting a confidence level of 95% and a margin of error of 5%. Thus, this empirical study provides a particular motivation for further qualitative investigation of review comments to acquire in-depth knowledge on the influence of reviewing on refactoring-inducing PRs.

It is not suitable to generalize the conclusions, except when considering other OSS projects that follow a geographically distributed development and are aligned to "the Apache way" principles [3]. Thus, our findings are exclusively extended to cases that have common characteristics with Apache's projects.

7 RELATED WORK

By exploring the motivations and challenges of MCR, Bacchelli and Bird identified the code improvements as one of the objectives of reviewing [18]. A finding confirmed by subsequent study on convergent practices of code review by Rigby and Bird [66], Beller et al. [21], and MacLeod et al. [50]. Those findings support us in exploring refactorings as a relevant contribution from code reviewing.

The analysis of the technical aspects of code reviewing has been the focus of several empirical studies, in which a few measures have been considered: the number of reviewers by Jiang et al. [43], the review comments by Rigby and Bird [66] and by Beller et al. [21], the time to merge by Izquierdo-Cortazar [41], and the size of change by Baysal et al. [20]. They provided the first insights on code reviewing aspects investigated in our study. Also, studies explored the factors influencing code review quality. Bosu et al. discovered that changes' properties affect the review comments usefulness [25]. Nevertheless, Kononenko et al. carried out an analysis concerning how developers perceive code review quality [45], and figured out that the thoroughness of feedback is the main influencing factor to code review quality. Those results corroborate with the findings on the technical aspects empirically studied in [20, 21, 43, 66], thus constituting an enriched set of technical aspects for investigation.

Paixão et al. found that refactorings' motivations may emerge from code review and influence the composition of edits and number of reviews by analyzing Gerrit reviews [17]. These findings inspired us towards expanding the knowledge regarding code review aspects in GitHub refactoring-inducing PRs. Pantiuchina et al. analyzed discussion and commits of merged PRs, containing at least one refactoring in one of their commits, and found that most refactorings are triggered from either the original intents of PRs or discussion [61]. Those findings are motivating since they indicate the influence that review, at the PR level, has on refactorings. Our study differs from those previous ones because we distinguished refactoring-inducing PRs from non-refactoring-inducing PRs by exploring both reviewing-related aspects and refactoring-inducement.

8 CONCLUDING REMARKS

We investigated technical aspects characterizing refactoring-inducing PRs based on data mined from GitHub and refactorings detected by RefactoringMiner. Our results reveal significant differences between refactoring-inducing and non-refactoring-inducing PRs, and a substantial number refactoring edits induced by code reviewing. As future work, we suggest (i) a further investigation of review comments aiming to identify patterns/practices that could indicate the refactoring-inducement as a contribution of the code review process to the code submitted within PRs; and (ii) exploration of human aspects of reviewers, aiming to enhance the understanding of refactoring-inducement at the PR level. Replications also are highly welcome, since they can support the elaboration of a theory on refactoring-inducing PRs.

ACKNOWLEDGMENTS

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).

REFERENCES

[1] 2001. Manifesto for Agile Software Development. https://agilemanifesto.org/. Accessed on: August 2020.

[2] 2019. The Apache Software Foundation Expands Infrastructure with GitHub Integration. https://t.ly/amPK. Accessed on: June 2020.

[3] 2019. Briefing: The Apache Way. https://www.apache.org/theapacheway/. Accessed on: June 2020.

[4] 2020. The 2020 State of the Octoverse – GitHub Report. https://octoverse.github.com/. Accessed on: May 2021.

[5] 2020. The Apache Software Foundation Projects Statistics. https://t.ly/DpAU. Accessed on: November 2020.

[6] 2020. GitHub Developer Guide GraphQL API v4. https://developer.github.com/v4/. Accessed on: June 2020.

[7] 2020. GitHub Developer Guide REST API v3. https://developer.github.com/v3/. Accessed on: June 2020.

[8] 2020. GitHub Platform. https://github.com. Accessed on: November 2020.

[9] 2020. GitHub Pull Requests. https://git.io/JILTS. Accessed on: June 2020.

[10] 2021. An Exploratory Study on Refactoring-Inducing Pull Requests – Reproduction Kit. https://doi.org/10.5281/zenodo.5106106.

[11] 2021. RefactoringMiner – A Refactoring Detection Tool. https://github.com/tsantalis/RefactoringMiner. Accessed on: September 2019.

[12] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. 1993. Mining Association Rules between Sets of Items in Large Databases. *ACM SIGMOD Record* 22, 2 (June 1993), 207–216. https://doi.org/10.1145/170036.170072

[13] Everton L. G. Alves, Myoungkyu Song, and Miryung Kim. 2014. RefDistiller: A Refactoring Aware Code Review Tool for Inspecting Manual Refactoring Edits. In *22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Hong Kong, China). 751–754. https://doi.org/10.1145/2635868.2661674

[14] Everton L. G. Alves, Myoungkyu Song, Tiago Massoni, Patricia D. L. Machado, and Miryung Kim. 2018. Refactoring Inspection Support for Manual Refactoring Edits. *IEEE Transactions on Software Engineering* 44, 4 (2018), 365–383. https://doi.org/10.1109/TSE.2017.2679742

[15] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. 1999. OPTICS: Ordering Points to Identify the Clustering Structure. In *1999 ACM SIGMOD International Conference on Management of Data*. Philadelphia, USA, 49–60. https://doi.org/10.1145/304182.304187

[16] Howard Anton and Chris Rorres. 2014. *Elementary Linear Algebra: Applications Version* (eleventh ed.). Wiley.

[17] M. Paix ao, A. Uchôa, A. C. Bibiano, D. Oliveira, A. Garcia, J. Krinke, and E. Arvonio. 2020. *Behind the Intents: An In-Depth Empirical Study on Software Refactoring in Modern Code Review*. ACM, New York, NY, USA, 125–136.

[18] Alberto Bacchelli and Christian Bird. 2013. Expectations, Outcomes, and Challenges of Modern Code Review. In *35th International Conference on Software Engineering*. San Francisco, USA, 712–721. https://doi.org/10.1109/ICSE.2013.6606617

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

[20] Olga Baysal, Oleksii Kononenko, Reid Holmes, and Michael W. Godfrey. 2016. Investigating Technical and Non-Technical Factors Influencing Modern Code Review. *Empirical Software Engineering* 21, 3 (June 2016), 932–959. https://doi.org/10.1007/s10664-015-9366-8

[21] Moritz Beller, Alberto Bacchelli, Andy Zaidman, and Elmar Juergens. 2014. Modern Code Reviews in Open-Source Projects: Which Problems Do They Fix?. In *11th Working Conference on Mining Software Repositories*. Hyderabad, India, 202–211. https://doi.org/10.1145/2597073.2597082

[22] Fernando Berzal, Ignacio Blanco, Daniel Sánchez, and María-Amparo Vila. 2002. Measuring the Accuracy and Interest of Association rRules: A New Framework. *Intelligent Data Analysis* 6, 3 (Aug. 2002), 221–235. https://doi.org/10.3233/IDA-2002-6303

[23] Giuseppe Bonaccorso. 2017. *Machine Learning Algorithms* (1 ed.). Packt Publishing.

[24] Amiangshu Bosu, Jeffrey C. Carver, Christian Bird, Jonathan Orbeck, and Christopher Chockley. 2017. Process Aspects and Social Dynamics of Contemporary Code Review: Insights from Open Source Development and Industrial Practice at Microsoft. *IEEE Transactions on Software Engineering* 43, 1 (Jan. 2017), 56–75. https://doi.org/10.1109/TSE.2016.2576451

[25] Amiangshu Bosu, Michaela Greiler, and Christian Bird. 2015. Characteristics of Useful Code Reviews: An Empirical Study at Microsoft. In *12th Working Conference on Mining Software Repositories*. Florence, Italy, 146–156. https://doi.org/10.1109/MSR.2015.21

[26] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur. 1997. Dynamic Itemset Counting and Implication Rules for Market Basket Data. In *1997 ACM SIGMOD International Conference on Management of Data* (Tucson, USA). New York, NY, USA, 255–264. https://doi.org/10.1145/253260.253325

[27] Neil Burdess. 2010. *Starting Statistics: a Short, Clear Guide*. SAGE Los Angeles. 187 pages.

[28] M. Emre Celebi and Kemal Aydin. 2016. *Unsupervised Learning Algorithms* (1st ed.). Springer Publishing Company, Incorporated.

[29] Scott Chacon and Ben Straub. 2014. *Pro Git* (2nd ed.). Apress, USA.

[30] Frans Coenen, Graham Goulbourne, and Paul Leng. 2004. Tree Structures for Mining Association Rules. *Data Mining and Knowledge Discovery* 8, 1 (Jan. 2004), 25–51. https://doi.org/10.1023/B:DAMI.0000005257.93780.3b

[31] Bradley Efron and Robert J Tibshirani. 1993. *An Introduction to the Bootstrap*. Chapman and Hall, London, England.

[32] Michael E. Fagan. 1976. Design and Code Inspections to Reduce Errors in Program Development. *IBM Systems Journal* 15, 3 (1976), 182–211. https://doi.org/10.1147/sj.153.0182

[33] Martin Fowler. 1999. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman Publishing Co., Inc., USA.

[34] Xi Ge, Saurabh Sarkar, and Emerson Murphy-Hill. 2014. Towards Refactoring-aware Code Review. In *7th International Workshop on Cooperative and Human Aspects of Software Engineering* (Hyderabad, India). 99–102. https://doi.org/10.1145/2593702.2593706

[35] Xi Ge, Saurabh Sarkar, Jim Witschey, and Emerson Murphy-Hill. 2017. Refactoring-Aware Code Review. In *2017 Symposium on Visual Languages and Human-Centric Computing (VL/HCC'17)*. Raleigh, USA, 71–79. https://doi.org/10.1109/VLHCC.2017.8103453

[36] Liqiang Geng and Howard J. Hamilton. 2006. Interestingness Measures for Data Mining: A Survey. *Comput. Surveys* 38, 3 (Sept. 2006), 9–es. https://doi.org/10.1145/1132960.1132963

[37] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An Exploratory Study of the Pull-Based Software Development Model. In *36th International Conference on Software Engineering*. Hyderabad, India, 345–355. https://doi.org/10.1145/2568225.2568260

[38] Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. 2016. Work Practices and Challenges in Pull-Based Development: The Contributor's Perspective. In *38th International Conference on Software Engineering*. Austin, USA, 285–296. https://doi.org/10.1145/2884781.2884826

[39] Jiawei Han, Jian Pei, and Yiwen Yin. 2000. Mining Frequent Patterns without Candidate Generation. *SIGMOD Record* 29, 2 (May 2000), 1–12. https://doi.org/10.1145/335191.335372

[40] Péter Hegedüs, István Kádár, Rudolf Ferenc, and Tibor Gyimóthy. 2018. Empirical Evaluation of Software Maintainability Based on a Manually Validated Refactoring Dataset. *Information and Software Technology* 95 (2018), 313–327. https://doi.org/10.1016/j.infsof.2017.11.012

[41] Daniel Izquierdo-Cortazar, Lars Kurth, Jesus M. Gonzalez-Barahona, Santiago Dueñas, and Nelson Sekitoleko. 2016. Characterization of the Xen Project Code Review Process: An Experience Report. In *13th International Conference on Mining Software Repositories*. Austin, USA, 386–390. https://doi.org/10.1145/2901739.2901778

[42] Tao Ji, Liqian Chen, Xin Yi, and Xiaoguang Mao. 2020. Understanding Merge Conflicts and Resolutions in Git Rebases. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. 70–80. https://doi.org/10.1109/ISSRE5003.2020.00016

[43] Yujuan Jiang, Bram Adams, and Daniel M. German. 2013. Will My Patch Make It? And How Fast?: Case Study on the Linux Kernel. In *10th Working Conference on Mining Software Repositories* (San Francisco, USA). 101–110.

[44] Yoshio Kataoka, Takeo Imai, Hiroki Andou, and Tetsuji Fukaya. 2002. A Quantitative Evaluation of Maintainability Enhancement by Refactoring. In *2002 International Conference on Software Maintenance*. USA, 576–585. https://doi.org/10.1109/ICSM.2002.1167822

[45] Oleksii Kononenko, Olga Baysal, and Michael W. Godfrey. 2016. Code Review Quality: How Developers See It. In *38th International Conference on Software Engineering*. Austin, EUA, 1028–1038. https://doi.org/10.1145/2884781.2884840

[46] Oleksii Kononenko, Tresa Rose, Olga Baysal, Michael Godfrey, Dennis Theisen, and Bart de Water. 2018. Studying Pull Request Merges: A Case Study of Shopify's Active Merchant. In *40th International Conference on Software Engineering: Software Engineering in Practice*. Gothenburg, Sweden, 124–133. https://doi.org/10.1145/3183519.3183542

[47] C. Lebeuf, M. Storey, and A. Zagalsky. 2018. Software Bots. *IEEE Software* 35, 01 (Jan. 2018), 18–23. https://doi.org/10.1109/MS.2017.4541027

[48] Gwendolyn K. Lee and Robert E. Cole. 2003. From a Firm-Based to a Community-Based Model of Knowledge Creation: The Case of the Linux Kernel Development. *Organization Science* 14, 6 (2003), 633–649. https://doi.org/10.1287/orsc.14.6.633.

24866

[49] Zhi-Xing Li, Yue Yu, Gang Yin, Tao Wang, and Huai-Min Wang. 2017. What are They Talking about? Analyzing Code Reviews in Pull-Based Development Model. *Journal of Computer Science and Technology* 32, 6 (Nov. 2017), 1060–1075. https://doi.org/10.1007/s11390-017-1783-2

[50] Laura MacLeod, Michaela Greiler, Margaret-Anne Storey, Christian Bird, and Jacek Czerwonka. 2018. Code Reviewing in the Trenches: Challenges and Best Practices. *IEEE Software* 35, 04 (Jul. 2018), 34–42. https://doi.org/10.1109/MS.2017.265100500

[51] Hassan Mansour and Nikolaos Tsantalis. 2020. Refactoring Aware Commit Review Chrome Extension. https://t.ly/J3Wr. Accessed on: November, 2020.

[52] Martin N Marshall. 1996. Sampling for Qualitative Research. *Family Practice* 13, 6 (Dec. 1996), 522–526. https://doi.org/10.1093/fampra/13.6.522

[53] Kenneth O. McGraw and Seok P. Wong. 1992. A Common Language Effect Size Statistic. *Psychological Bulletin* 111, 2 (1992), 361–365. https://doi.org/10.1037/0033-2909.111.2.361

[54] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan. 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 *11th Working Conference on Mining Software Repositories*. ACM, Hyderabad, India, 192–201. https://doi.org/10.1145/2597073.2597076

[55] Ehsan Mirsaeedi and Peter C. Rigby. 2020. Mitigating Turnover with Code Review Recommendation: Balancing Expertise, Workload, and Knowledge Distribution. In *ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE'20)*. ACM, 1183–1195. https://doi.org/10.1145/3377811.3380335

[56] William F. Opdyke. 1992. *Refactoring Object-Oriented Frameworks*. Ph.D. Dissertation. Department of Computer Science, University of Illinois at Urbana-Champaign. UMI Order No. GAX93-05645.

[57] William F. Opdyke and Ralph E. Johnson. 1990. Refactoring: An Aid in Designing Application Frameworks and Evolving Object-Oriented Systems. In *Proceedings of the Symposium on Object Oriented Programming Emphasizing Practical Applications*. New York, USA.

[58] Matheus Paixão, Jens Krinke, DongGyun Han, Chaiyong Ragkhitwetsagul, and Mark Harman. 2019. The Impact of Code Review on Architectural Changes. *IEEE Transactions on Software Engineering* (2019), 1–1. https://doi.org/10.1109/TSE.2019.2912113

[59] Matheus Paixão and Paulo H. Maia. 2019. Rebasing in Code Review Considered Harmful: A Large-Scale Empirical Investigation. In *2019 19th International Working Conference on Source Code Analysis and Manipulation*. 45–55. https://doi.org/10.1109/SCAM.2019.00014

[60] Fabio Palomba, Andy Zaidman, Rocco Oliveto, and Andrea De Lucia. 2017. An Exploratory Study on the Relationship between Changes and Refactoring. In *25th International Conference on Program Comprehension*. Buenos Aires, Argentina, 176–185. https://doi.org/10.1109/ICPC.2017.38

[61] Jevgenija Pantiuchina, Fiorella Zampetti, Simone Scalabrino, Valentina Piantadosi, Rocco Oliveto, Gabriele Bavota, and Massimiliano Di Penta. 2020. Why Developers Refactor Source Code: A Mining-Based Study. *ACM Transactions on Software Engineering Methodology* 29, 4, Article 29 (Sept. 2020), 30 pages. https://doi.org/10.1145/3408302

[62] Luca Pascarella, Davide Spadini, Fabio Palomba, Magiel Bruntink, and Alberto Bacchelli. 2018. Information Needs in Contemporary Code Review. *Proceedings of the ACM on Human-Computer Interaction* 2, CSCW, Article 135 (Nov. 2018), 27 pages. https://doi.org/10.1145/3274404

[63] Achyudh Ram, Anand Ashok Sawant, Marco Castelluccio, and Alberto Bacchelli. 2018. What Makes a Code Change Easier to Review: An Empirical Investigation on Code Change Reviewability. In *26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Lake Buena Vista, USA, 201–212. https://doi.org/10.1145/3236024.3236080

[64] Sebastian Raschka. 2018. MLxtend: Providing Machine Learning and Data Science Utilities and Extensions to Python's Scientific Computing Stack. *Journal of Open Source Software* 3, 24 (2018), 638. https://doi.org/10.21105/joss.00638

[65] Peter Rigby, Brendan Cleary, Frederic Painchaud, Margaret-Anne Storey, and Daniel German. 2012. Contemporary Peer Review in Action: Lessons from Open Source Development. *IEEE Software* 29, 6 (Nov. 2012), 56–61. https://doi.org/10.1109/MS.2012.24

[66] Peter C. Rigby and Christian Bird. 2013. Convergent Contemporary Software Peer Review Practices. In *9th Joint Meeting on Foundations of Software Engineering*. Saint Petersburg, Russia, 202–212. https://doi.org/10.1145/2491411.2491444

[67] Peter C. Rigby, Daniel M. German, Laura Cowen, and Margaret-Anne Storey. 2014. Peer Review on Open-Source Software Projects: Parameters, Statistical Models, and Theory. *ACM Transactions on Software Engineering Methodology* 23, 4, Article 35 (Sept. 2014), 33 pages. https://doi.org/10.1145/2594458

[68] Peter C. Rigby, Daniel M. German, and Margaret-Anne Storey. 2008. Open Source Software Peer Review Practices: A Case Study of the Apache Server. In *30th International Conference on Software Engineering*. ACM, Leipzig, Germany, 541–550. https://doi.org/10.1145/1368088.1368162

[69] Romain Robbes and Michele Lanza. 2007. Characterizing and Understanding Development Sessions. In *15th IEEE International Conference on Program Comprehension*. USA, 155–166. https://doi.org/10.1109/ICPC.2007.12

[70] Per Runeson and Martin Höst. 2009. Guidelines for Conducting and Reporting Case Study Research in Software Engineering. *Empirical Software Engineering* 14, 2 (April 2009), 131–164. https://doi.org/10.1007/s10664-008-9102-8

[71] Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern Code Review: A Case Study at Google. In *40th International Conference on Software Engineering: Software Engineering in Practice*. Gothenburg, Sweden, 181–190. https://doi.org/10.1145/3183519.3183525

[72] Chris Sauer, D. Ross Jeffery, Lesley Land, and Philip Yetton. 2000. The Effectiveness of Software Development Technical Reviews: A Behaviorally Motivated Program of Research. *IEEE Transactions on Software Engineering* 26, 1 (Jan. 2000), 1–14. https://doi.org/10.1109/32.825763

[73] Margaret-Anne Storey and Alexey Zagalsky. 2016. Disrupting Developer Productivity One Bot at a Time. In *24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Seattle, WA, USA). New York, NY, USA, 928–931. https://doi.org/10.1145/2950290.2983989

[74] Gábor Szőke, Csaba Nagy, Rudolf Ferenc, and Tibor Gyimóthy. 2014. A Case Study of Refactoring Large-Scale Industrial Systems to Efficiently Improve Source Code Quality. In *2014 International Conference on Computational Science and its Applications (ICCSA'14)*. Springer International Publishing, Guimarães, Portugal, 524–540. https://doi.org/10.1007/978-3-319-09156-3_37

[75] Yida Tao, Yingnong Dang, Tao Xie, Dongmei Zhang, and Sunghun Kim. 2012. How Do Software Engineers Understand Code Changes? An Exploratory Study in Industry. In *20th ACM SIGSOFT International Symposium on the Foundations of Software Engineering*. Cary, USA, Article 51, 11 pages. https://doi.org/10.1145/2393596.2393656

[76] Patanamon Thongtanunam, Shane Mcintosh, Ahmed E. Hassan, and Hajimu Iida. 2017. Review Participation in Modern Code Review. *Empirical Software Engineering* 22, 2 (April 2017), 768–817. https://doi.org/10.1007/s10664-016-9452-6

[77] Nikolaos Tsantalis, Ameya Ketkar, and Danny Dig. 2020. RefactoringMiner 2.0. *IEEE Transactions on Software Engineering* (2020), 21 pages. https://doi.org/10.1109/TSE.2020.3007722

[78] Nikolaos Tsantalis, Matin Mansouri, Laleh M. Eshkevari, Davood Mazinanian, and Danny Dig. 2018. Accurate and Efficient Refactoring Detection in Commit History. In *40th International Conference on Software Engineering*. ACM, Gothenburg, Sweden, 483–494. https://doi.org/10.1145/3180155.3180206

[79] Dongkuan Xu and Yingjie Tian. 2015. A Comprehensive Survey of Clustering Algorithms. *Annals of Data Science* 2 (08 2015), 165–193. https://doi.org/10.1007/s40745-015-0040-1

[80] Alice Zheng and Amanda Casari. 2018. *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists* (1st ed.). O'Reilly Media, Inc.

[81] Jiaxin Zhu, Minghui Zhou, and Audris Mockus. 2016. Effectiveness of Code Contribution: From Patch-Based to Pull-Request-Based Tools. In *24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. Seattle, USA, 871–882. https://doi.org/10.1145/2950290.2950364