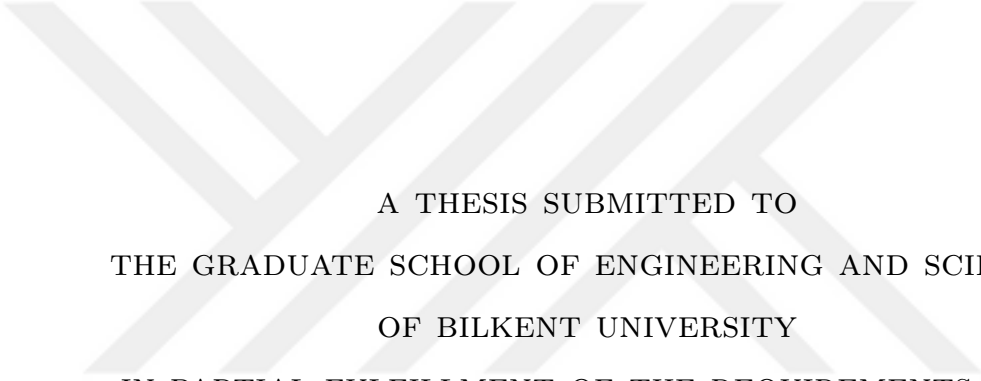


ANALYZING THE IMPACT OF PROCESS SMELLS IN CODE REVIEW



A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

By
Shirin Pirouzkhah
July 2022

Analyzing the Impact Of Process Smells In Code Review

By Shirin Pirouzkhah

July 2022

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



Eray Tüzün(Advisor)

Halil Altay Güvenir

İsmail Sengör Altıngövde

Approved for the Graduate School of Engineering and Science:

Orhan Arıkan
Director of the Graduate School

ABSTRACT

ANALYZING THE IMPACT OF PROCESS SMELLS IN CODE REVIEW

Shirin Pirouzkhah
M.S. in Computer Engineering
Advisor: Eray Tüzün
July 2022

Code review is a valuable modern software development practice for identifying potential problems in changesets and improving source code quality in software projects. Developers adhere to specific rules and best practices in an ideal code review process. Deviations from these best practices can potentially lead to unintended consequences. These deviations, also known as code review process smells, affect different parts of the code review process and its outcomes. This research empirically analyzes the correlations between process smells, and the impact of code review process smells (shared knowledge, reopened issues, time to complete review (TTCR), abandoned pull requests (PRs), number or review comments, and contribution of large changeset reviewers). The relationship between smells and their impact is correlation-based, but a causal relationship is not guaranteed because proving such relationships is not straightforward. To quantitatively analyze the impact of smells in the code review process, we investigated 244,903 PRs from four open source software (OSS) projects from Gerrit (Qt, Eclipse, Wireshark, and LibreOffice). As a result, we discovered that the existence of a committers' side smell in a pull request (PR) could lead to the occurrence of reviewers' side smells in the same PR. We also observed that reviewers assigned repeatedly by the same author to review the PR have a heavier review workload. Furthermore, the ping pong reviews and large changesets are most likely to increase the bug reopen ratio. In addition, we noticed that the time to complete review of a smelly PR is likely to be more than time to complete review of non-smelly PR. Also, the existence of process smells in a PR effect the number of review comments in that pull request.

Keywords: modern code review, process smells, code review anti-patterns, impacts of anti-patterns, impact of process smells.

ÖZET

KOD INCELEME SÜRECİNDE UYGUNSUZLUK ETKİSİNİN ANALİZİ

Shirin Pirouzkhah
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Danışmanı: Eray Tüzün
Temmuz 2022

Kod değerlendirme, modern yazılım geliştirme sırasında yapılan değişikliklerde çıkabilecek potansiyel problemleri saptama ve kaynak kodunu geliştirmede oldukça önemli bir aşamadır. Geliştiriciler ideal kod inceleme sürecinde birtakım kurallara bağlı kalmaktadırlar. Bu kurallardan sapmalar, kod değerlendirme uygunsuzlukları (smells) olarak da bilinir ve kod değerlendirmenin çıktılarını doğrudan etkiler. Bu araştırma, deneysel olarak kod değerlendirme sürecindeki uygunsuzlukları ile kod değerlendirme etkisi arasında ilişkiyi (paylaşılan bilgi, tekrar açılan konular, değerlendirme sürecini tamamlamak için geçen süre, terk edilmiş çekme istekleri (PR), değerlendirme yorumlarının sayısı, büyük ölçekli değişikliklerin değerlendirmeye katkısı). kod değerlendirme uygunsuzlukları ve uygunsuzlukların etkileri arasındaki ilişki korelasyon tabanlıdır, fakat neden tabanlı bir ilişki garanti değildir. Uygunsuzlukların kod değerlendirmeye etkisini sayısal bir şekilde analiz edebilmek için, Gerrit'ten dört açık kaynak kodlu projesinden (Qt, Eclipse, Wireshark ve LibreOffice) 244,903 PR inceledik. Sonuç olarak, çekme isteklerinde önerme-tarafı (commit) uygunsuzlukların varlığını keşfettik ve bu uygunsuzlukların değerlendirme-tarafı uygunsuzluklara sebep olduğunu gördük. Ayrıca değerlendirme aşamalarında tekrarlanan bu tür uygunsuzlukların değerlendirme sürecinde daha ağır iş yüklerine sebep olduğunu açığa çıkardık. Bunlara ek olarak, dört proje için de, pinpon değerlendirmelerin ve büyük ölçekli değişikliklerin sonrasında hata-tabanlı konuların tekrar açılmasının daha olası olduğunu gördük. Uygunsuz değerlendirmelerin tamamlanma sürelerinin uygunlara göre daha çok olduğunu fark ettik. Ayrıca, bir PR'deki süreç uygunsuzluklarının varlığı, o çekme talebindeki inceleme yorumlarının sayısını etkiler.

Anahtar sözcükler: modern kod değerlendirme, süreç uygunsuzlukları, kod

değerlendirme anti-desenleri, anti-desen etkileri, süreç uygunsuzluklarının etkisi.



Acknowledgement

First and foremost, I would like to express my genuine appreciation to my advisor Prof. Eray Tüzün for his support. His positive energy, work management, and work ethic always inspired me throughout my M.Sc. study.

I want to thank my family, especially my parents, Mohammadreza and Masoumeh, for their unconditional support, kindness, and unlimited love. I want to thank my sisters, Simin and Mina as well.

I would like to thank my friends, Elmira khajei, Ebru Buhara, Mahan Masboughi, Ali Kaan Sünnetçioğlu, and Elshen Asadi, that were always there for me on my good and bad days. I did not find much opportunity to find good friends in Bilkent due to coronavirus pandemic in the first two years. However, the third year was the most active and happiest time.

Contents

1	Introduction	1
1.1	Research Problem	2
1.2	Contributions	3
2	Background and Related Work	4
2.1	Code Review Process Smells	4
2.2	Related Work	6
3	Methodology	9
3.1	Bypassed Negative Reviewer Smell	9
3.2	Missing Context Smell	12
3.3	Review Buddies Smell	15
3.4	Categorizing Code Review Process Smells	17
3.5	Correlations Between Code Review Process Smells	19
3.6	Impact Of Code Review Process Smells	20

3.6.1	Shared knowledge impact	21
3.6.2	Reopened issues impact	23
3.6.3	Time to complete review impact	26
3.6.4	Abandoned pull requests impact	26
3.6.5	Number of review comments impact	27
3.6.6	Contribution of large changeset reviewers impact	28
4	Results	31
4.1	Code Review and Issue Tracking Datasets	31
4.2	Data Preprocessing	32
4.3	Correlation Results	34
4.3.1	Bypassed negative reviewer smell vs. sleeping review smell and ping pong smell	34
4.3.2	Large changeset smell (LCHS) vs sleeping review smell and ping pong smell	36
4.3.3	Missing context smell vs. sleeping review smell and ping pong smell	38
4.4	Impact Results	42
4.4.1	Shared knowledge impact	42
4.4.2	Reopened issues impact	45
4.4.3	Time to complete review (TTCR) impact	47

4.4.4	Abandoned pull requests impact	53
4.4.5	Number of review comments impact	54
4.4.6	Contribution of large changeset reviewers impact	58
5	Hypothesis Testing	60
5.1	Significance of Results in Correlation Between Smells, Reopened Issues Impact, and Abandoned Pull Requests Impact	61
5.2	TTCR, Review Workload, and Number of Review Comments VS. Process Smells	67
5.3	Correlation of A PR Being Smelly VS. Reopened Issues Impact and Abandoned Pull Requests Impact	72
5.4	Final Deduction	75
6	Discussion	77
6.1	Answers to Research Questions	77
6.2	Implications to Practitioners and Researchers	79
7	Threats to Validity	81
8	Conclusion and Future Work	84

List of Figures

3.1	Bypassed negative reviewer smell scheme	10
3.2	Pull request lifecycle and start point of each code review process smell	17
3.3	Correlations between code review process smells and their impacts	19
3.4	Scheme of shared knowledge impact detection algorithm in review buddies smell	22
3.5	Lifecycle of an issue in QT project issue tracking tool (Jira) . . .	24
3.6	Lifecycle of an issue in Eclipse, LibreOffice and Wireshark projects issue tracking tool (Bugzilla)	25
3.7	Scheme of algorithm to detect the pattern of contribution of large changeset reviewers	30
4.1	Double-positive (blue color), double-negative (green color), [single- positive and single-negative] (red color) ,and [single-negative and single-positive] (yellow color) cases in correlation between different smells	41
4.2	Frequency distribution of author-reviewer pairs in four Gerrit projects	42

4.3	Distribution of review workload (CBCS score, reviewed LOCs by reviewers of pairs) in two different groups, review buddy pairs, and non-review buddy pairs	44
4.4	Distribution of frequency and workload of high-frequency pairs (blue dots as review buddies) and low-frequency pairs (orange dots as non-review buddies)	45
4.5	Distribution of time to complete review of smelly and non-smelly PRs for five types of smells in Qt project	49
4.6	Distribution of time to complete review of smelly and non-smelly PRs for five types of smells in Eclipse project	50
4.7	Distribution of time to complete review of smelly and non-smelly PRs for five types of smells in Wireshark project	51
4.8	Distribution of time to complete review of smelly and non-smelly PRs for five types of smells in LibreOffice project	52
4.9	Distribution of number of review comments of smelly and non-smelly PRs for six types of smells in Qt project	55
4.10	Distribution of number of review comments of smelly and non-smelly PRs for six types of smells in Eclipse project	56
4.11	Distribution of number of review comments of smelly and non-smelly PRs for six types of smells in Wireshark project	57
4.12	Distribution of number of review comments of smelly and non-smelly PRs for six types of smells in LibreOffice project	58
4.13	Pattern of large changeset reviewers' contribution (ratio of reviewed PRs with large changesets) over time in four Gerrit projects	59

5.1	Correlations between code review process smells and their impacts (solid blue lines :confirmed by statistical tests, dashed red lines: rejected by statistical tests)	75
-----	---	----



List of Tables

3.1	Categorizing code review process smells	18
4.1	Details regarding the code review data of four Gerrit projects used	32
4.2	Details regarding the issue tracking data of four Gerrit projects used.	32
4.3	Bypassed negative reviewer smell frequency in four Gerrit projects	35
4.4	Correlations between bypassed negative reviewer smell vs sleeping review smell and ping pong smell and correlation percentage (Corr. perc.%)in four Gerrit projects	35
4.5	Correlations between large changeset smell (LCHS) vs sleeping review smell and ping pong smell in four Gerrit projects	36
4.6	The average number of iterations in the review and feedback cycle of PRs in different changeset (CHS) size categories in four Gerrit projects	37
4.7	Missing context smell frequency in four Gerrit projects	38
4.8	Correlations between missing context smell vs sleeping review smell and ping pong smell in four Gerrit projects	39

4.9	Total number of author-reviewer pairs and review buddies smell in Gerrit projects.	43
4.10	Average of LOCs reviewed by reviewers of high frequent pairs (review buddies) and low frequent pairs (non-review buddies) in four Gerrit projects.	43
4.11	Distribution of the smelly PRs linked to reopened issues for bypassed negative reviewer smell, review buddies smell and lack of code review smell in four Gerrit projects	46
4.12	Distribution of the smelly PRs linked to reopened issues for ping pong smell, sleeping review smell and large changeset smell in four Gerrit projects	46
4.13	Average TTCR of smelly and non smelly PRs (in days)	48
4.14	Average TTCR of smelly and non smelly PRs (in days)	48
4.15	Abandoned pull requests impact of two smell categories: ping pong and sleeping review smells in four Gerrit projects	54
4.16	Abandoned pull requests impact of two smell categories: LCHS and missing context smells in four Gerrit projects	54
5.1	Frequency of PRs with or without bypassed negative reviewer smell and sleeping review smell in Eclipse project	65
5.2	P-values of Fisher's exact test for correlation between different smells in four Gerrit projects. Expected Results (Blue Color), Unexpected Results (Red Color)	65
5.3	Odds ratio of Fisher's exact test for correlation between different smells in four Gerrit projects. Expected Results (Blue Color), Unexpected Results (Red Color)	65

5.4	P-values of Fisher's exact test for reopen impact in four Gerrit projects. Expected Results (Blue Color), Unexpected Results (Red Color)	66
5.5	Odds ratio of Fisher's exact test for reopen impact in four Gerrit projects. Expected Results (Blue Color), Unexpected Results (Red Color)	66
5.6	P-values of Fisher's exact test for abandoned PRs impact in four Gerrit projects. Expected Results (Blue Color), Unexpected Results (Red Color)	66
5.7	Odds ratio of Fisher's exact test for abandoned PRs impact in four Gerrit projects. Expected Results (Blue Color), Unexpected Results (Red Color)	66
5.8	P-value and effect size (r) of statistical test with Mann-Whitney U testing for shared knowledge impact, workload of review buddies and non-review buddies in four Gerrit projects. Expected Results (Blue Color)	70
5.9	P-value and effect size (r) of statistical test with Mann-Whitney U testing for TTCR of smelly and non smelly PRs in four Gerrit projects. Expected Results (Blue Color)	71
5.10	P-value and effect size (r) of statistical test with Mann-Whitney U testing for number of review comments of smelly and non smelly PRs in four Gerrit projects. Expected Results (Blue Color), Unexpected Results (Red Color)	71
5.11	An example of data used in chi-square test to analyze the significance of relationship between a PR being smelly and a PR being linked to reopen issues resulted (observed) in Eclipse project . . .	73

5.12 χ^2 and p-values of chi-square test for reopened issues impact in
four Gerrit projects ($df = 5$). Expected Results (Blue Color), Un-
expected Results (Red Color) 73

5.13 χ^2 and p-values of chi-square test for abandoned PRs impact in
four Gerrit projects ($df = 3$). Expected Results (Blue Color), Un-
expected Results (Red Color) 73



Chapter 1

Introduction

Code review is a commonly used process in software development projects that aims to improve the quality of the source code. Ideally, having a proper code review process improves knowledge transfer and code ownership and reduces the likelihood of bug reopens. Previous work has investigated deviations from an optimal code review process, known as code review process smells, and their possible causes. Dogan and Tuzun [1] proposed a taxonomy of seven code review process smells by conducting an MLR (Multi-vocal Literature Review) and evaluated the review histories (software development evolution histories) of open source projects.

In this study, we empirically investigate the correlations and impacts of these process smells by building on this taxonomy. Code review process smells - deviations from ideal code review practices- are essential in terms of their effect on different aspects. Code review process smells are categorized as committers' side smells (initiated by the committer of a changeset), and reviewers' side smells (initiated by reviewers of a changeset) based on the initial point that smell has originated, as well as who has initiated the smell. These smells can be interdependent as a committers' side smell can cause the emergence of a reviewers' side smells in the same pull request (pull request level). Furthermore, the occurrence

of one smell may create problems on the project level and cause negative impacts on various aspects of the project. Despite previous work investigating the impact of code smells on maintenance effort [2], merge conflicts [3], and energy efficiency [4], to the best of our knowledge, this is the first study to investigate the impact of code review process smells.

1.1 Research Problem

In this study, we aim to answer the following research questions:

RQ1- What are the correlations between code review process smells?

To answer the first question, we measure how the existence of one committers' side smell can lead to other reviewers' side smells and analyze the co-occurrence of these smells. By performing proper statistical tests on these co-occurrences, we confirm the correlation among different smells.

RQ2- What are the impacts of code review process smells, and how prevalent are these impacts in OSS projects?

Applying ideal code review practices to the software development process raises the expectation of observing benefits and associated characteristics, including improvements in knowledge transfer, a lower reopen ratio, a lower ratio of abandoned pull requests, and higher contributions of reviewers. By analyzing and measuring these characteristics in the presence of code review process smells, we propose six impacts of code review process smells that negatively affect various aspects of project. We empirically analyze each impact by evaluating the code review histories of four Gerrit projects (Qt, Eclipse, Wireshark, and LibreOffice).

1.2 Contributions

The main contributions of this thesis are:

- Proposed one additional code review process smell (bypassed negative reviewer smell) and improvements in the detection algorithm of two other smells (review buddies smell and missing context smell).
- Performed an empirical analysis with four OSS projects to demonstrate correlations between smells and impacts of smells with varying significance.
- Observed that the existence of committers' side smells will likely lead to occurrence of reviewer's side smells.
- Observed that the existence of code review process smells emerge consequences (impacts) in different aspects.
- Performed statistical analysis of correlations between smells and impacts of smells.

In the following chapter, we provide a background on the code review process smells and the related work. In Chapter 3, we explain our methodology addressing one additional process smell, the changes we performed to the detection algorithms compared to Dogan and Tuzun [1], the correlations between code review process smells, and impacts of smells and the algorithms used to detect these impacts. In Chapter 4, we share the details of the datasets and present the preprocessing steps. We also illustrate the results of our empirical study of the correlations and impacts of code review process smells on four OSS projects, and in Chapter 5, we perform statistical tests to analyze the significance of our results quantitatively. In Chapter 6, we discuss the RQs of the study and its practical implications to practitioners and researchers. In Chapter 7, we discuss the threats to validity of our study. In Chapter 8, we present our conclusions and possible future works.

Chapter 2

Background and Related Work

In this chapter, we summarize the taxonomy of code review process smells based on a study conducted by Dogan and Tuzun [1]. Then we will summarize several studies on non-ideal code reviews and the factors that each study introduces as the main reason for the potential problems in the code review process in Section 2.2.

2.1 Code Review Process Smells

Dogan and Tuzun [1] introduced the taxonomy of following code review process smells. They also conducted a developer survey to observe the perception of 32 practitioners related to each code review process smell in their study. In the following, we list seven code review process smells, their definitions, and the developer survey results.

1- Lack of code review smell (LOCR): Lack of code review means that a merged pull request (PR) is not reviewed by a developer (non-reviewed) or is reviewed only by the authors themselves (self-reviewed). Based on the survey, 31 out of 32 practitioners think that lack of code review is a critical smell, but only four reported encountering it regularly. Possible root causes of this smell

are availability problems, high self-confidence, or time pressure.

2- Looks good to me smell (LGTM): LGTM refers to the phrase “looks good to me” in open source projects. This smell means developers may have performed inefficient code reviews with negligence without proper comments and feedback and directly approved the changeset with only a LGTM comment. Based on the survey, 31 out of 32 practitioners think that lack of code review is a critical smell, and 15 reported encountering it regularly. Possible root causes of occurrence of this smell are an irrelevant reviewer or reviewer’s negligence.

3- Reviewer-author ping-pong smell: This smell occurs when the number of feedback and modification iterations between the author and the reviewer is excessive. Reviews with more than three iterations of the review cycle are considered to have a ping pong smell. Based on the survey, only 17 out of 32 practitioners think that lack of code review is a critical smell, and 11 reported encountering it regularly. Possible root causes of the occurrence of this smell are that the reviewer cannot find all of the defects or the author cannot apply all the required changes.

4- Sleeping review smell: Sleeping review refers to code reviews that take longer than two days to be completed. Based on the survey, 26 out of 32 practitioners think that lack of code review is a critical smell, and 21 practitioners reported encountering it regularly. Possible root causes of this smell are availability problems, or the submitted changeset is difficult or large to be reviewed.

5-Large changesets smell: Large changesets refer to submitted PRs containing a large number of lines of code that have been changed by the author and need to be reviewed. Based on the survey, 28 out of 32 practitioners think that lack of code review is a critical smell, but only 11 reported encountering it regularly. Possible root causes of this smell are committer trying to complete

everything in a single changeset, or the task is large initially.

6- Missing context smell: This smell refers to PRs that neither have an adequately long description nor any link to an issue. Based on the survey, 28 out of 32 practitioners think that lack of code review is a critical smell, and 14 practitioners reported encountering it regularly. A possible root cause of this smell is the nonconformance of PR authors to the formal software development process.

7- Review buddies smell: If a reviewer has reviewed and evaluated at least half of the commits provided by an author, that reviewer is referred to be the author’s review buddy. Involved developers in this type of code review process smell are called ”buddies”. Based on the survey, 28 out of 32 practitioners think that lack of code review is a critical smell, and 21 practitioners reported encountering it regularly. A possible root cause of this smell is the convenience of picking the same reviewer.

The smell detection algorithm of the review buddies and missing context smells are modified and improved in our study. These modifications are described in Section 3.

2.2 Related Work

There are several previous studies on the results of unsuccessful code reviews. Most studies of non-ideal code reviews consider only a single factor or impact corresponding to a smell category in the Dogan and Tuzun taxonomy [1]. We classified previous studies based on their topics related to our study.

General factors of code review: Code churn, change description, and commit history are three critical factors that affect the reviewability of a changeset,

according to Ram and Bacchelli [5]. Baysal et al. [6] show that technical parameters like patch size and components, as well as non-technical elements like the reviewer and author experience, have a considerable impact on code review results. Chouchen et al. [7] stress the social side of the author-reviewer connection by categorizing the anti-patterns of modern code reviews into confused, divergent, low-involvement, shallow, and toxic code reviews. Pascarella et al. [8] explored how code reviews affect six types of suboptimal design decisions, known as code smells. They investigated how code review affects the severity of six types of code smells in seven Java open-source projects and found that active and participative code reviews significantly reduce code smell severity.

Behaviour of reviewers: Rigby and Storey [9] manually examined reviews across five OSS projects to look at the processes and actions developers take to identify code changes they may review. They found out that if reviewers are not interested in a PR, they would rather delay PR review and PR experiences a longer review time.

Divergent votes in code review process: Hirao et al. [10] discovered that 15%–37% of patches with multiple review scores suffer from divergent review votes when reviewers cannot have a consensus on the final vote due to divergent evaluations and votes. Divergent scores can slow down the code review process and increase the time to complete the review. Furthermore, Hirao et al. [11] observed that a reviewer with more experience will probably have a higher level of agreement than a reviewer with less expertise. Moreover, it is more likely that a review sent to a reviewer with a lower degree of the agreement would require more time for both reviewing and discussion.

Shared knowledge and reviewers' contribution: Bacchelli and Bird [12] believe that the development team's shared knowledge is one of the expectations of a successful modern code review. However, Caulo et al. [13] discovered that the transfer of knowledge to developers by receiving feedback from reviewers does not positively affect the quality of developer's future contributions.

Reopened issues: According to Rigby and German [14], in order to have a correct setup for code review practices, 80% of key team members will be involved in the process. Lack of code review is one of the bad practices that may happen due to low code review participation and may cause bugs to surface later on via

bug reopen. Using modern code review, according to Bacchelli and Bird [12], increases the quality of a change to a software system, reducing the future bug reopen rate. Zazworka et al. [15] believe that when the number of reopened bugs passes a particular threshold in a project, instability may increase in the software system. They investigate various reasons for bug reopening and the correlation between these reasons and bug reopening. They introduced various factors of bug reopen and categorized these factors ; (1) whether the bug was fixed or not fixed, (2) the source of the bug (e.g., human review or code analysis tool), and (3) organizational (team structure, temporary staff) and geographical factors. McIntosh et al. [16] present empirical evidence of the correlation between lack of code review coverage and an increase in post-release defects, with commits without code review coverage in three open-source repositories (OSS) having two to five more post-release defects. Bavota and Russo [17] discovered that commits that have not been reviewed are more than twice as likely to include errors as commits that have been reviewed.

Abandoned pull requests: Khatoonabandi et al. [18] investigated pull request abandonment reasons in GitHub projects. They found that long reviews, inexperienced contributors, and complicated PRs increase the likelihood of abandonment, and the rate of abandonment varies with the maturity or workload of the projects. They also observed that PRs with more than three iterations and those submitted by inexperienced contributors are more likely to get abandoned. Moreover, Li et al. [19] discovered 12 main reasons for pull request abandonment. Top-ranked reasons are lack of reviewers' response and committer experiencing lack of time.

Chapter 3

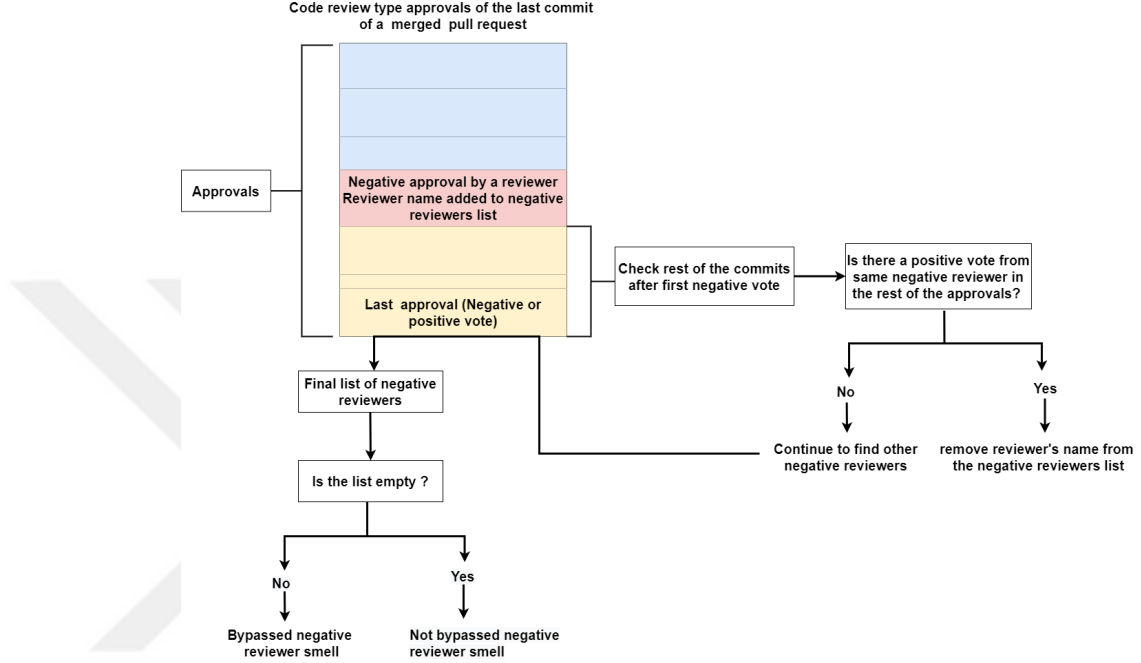
Methodology

In this chapter, to address the RQs, we identify one additional code review process smell that we define and evaluate (bypassed negative reviewer smell) and changes we applied to detection algorithms for missing context smell and review buddies smell. Furthermore, the correlations between smells are investigated. In addition, we categorize six impacts (shared knowledge, reopened issues, time to complete review, abandoned pull requests, number of review comments, and contribution of large changeset reviewers) which are potential side effects of code review process smells.

3.1 Bypassed Negative Reviewer Smell

Hirao et al. [10] illustrated that 15%–37% of patches with multiple review scores have divergent review votes, and 29% of patches with divergent scores are eventually abandoned. Divergent votes (positive and negative votes) may create contradiction and potential problems. We introduce the "bypassed negative reviewer" smell and analyze its impacts to exhibit the problem.

Figure 3.1: Bypassed negative reviewer smell scheme



Bypassed negative reviewer smell happens in PRs with divergent votes. This smell occurs when one of the reviewers of a PR completes the review with a negative vote, then the PR gets verified and merged despite the negative vote. In this kind of PRs, negative reviewers later do not change their vote to a positive vote. However, the PR still gets verified and merged. To investigate this smell, the following steps are performed:

1. Only "closed" PRs with "merged" status are considered since the definition of this smell is satisfied only in merged PRs, so abandoned PR are not considered.
2. Among all PRs, only the ones with at least one negative vote are extracted to retrieve the reviewers who voted negatively and get bypassed afterward. To check if a PR has a negative vote, we iterate over all of the approvals in the PR.

3. If there is a negative approval by a reviewer who does not have any subsequent positive approval, the PR is considered smelly since it has been merged despite the presence of a negative reviewer who has not turned their vote to a positive one.

This smell has several impacts, such as increased bug reopen rate, increased feedback-modification iterations in the code review of the same PR, increased time to complete review, and the change in the number of review comments.

Algorithm 1 shows pseudo-code for bypassed negative reviewer smell detection, and Figure 3.1 shows the scheme of detection algorithm of this smell through code review approvals of a merged PR. Note that in Gerrit projects, only the last commit and its approvals are essential because only the last commit of PRs are merged to the main codeline.

Algorithm 1 Bypassed negative reviewer smell detection algorithm

```

1: for each PR in review_history do
2:   neg_rev_list  $\leftarrow$  []
3:   if last_commit in PR has approvals then
4:     for each approval in Approvals do
5:       if approvalType is "Code-Review" and vote is negative then
6:         if approvalName exists then
7:           if approvalName is not in bots list then
8:             neg_rev_list.append(approvalName)
9:           if approvalType is "Code-Review" and vote is positive then
10:            if approvalName exists then
11:              if approvalName is in neg_rev_list then
12:                neg_rev_list.remove(approvalName)
13:          if neg_rev_list is not empty then
14:            return True
15:          else
16:            return False

```

3.2 Missing Context Smell

This smell means that a PR suffers from missing linkage to an associated issue or lacks proper description in the commit message [1]. Dogan and Tuzun [1] mined only code review data to check whether PRs contain information about linked issues (e.g., bug ID or bug number). Since we also utilize issue data of each project (data retrieved from issue tracking systems (IST) used by each project), we also mine this data to retrieve commits (commit hashes) reported in each issue. We check the linkage between PRs and issues from two sources; commit hashes reported in issue data and issue numbers reported in the commit message in the code review data. Algorithm 2 shows how we retrieve commit hashes of the PRs reported in issue data.

This smell has several impacts, such as increased feedback-modification iterations in the code review of the same PR, increased time to complete review, and decreased number of review comments impact.

Algorithm 2 Commit hashes reported in the issue data

```
1: if IssueData from Bugzilla then
2:   for each issue in IssueData do
3:     if "see_also" or URL in issue do
4:       commits_in_issueData  $\leftarrow$  issue["see_also"] or issue["URL"]
5: if IssueData from Jira then
6:   for each issue in IssueData do
7:     for each item in issue["changelog.history"] do
8:       if item["field"] = "Commits" then
9:         commits_in_issueData  $\leftarrow$  item
```

The projects that are chosen for our analysis use Jira and Bugzilla as their issue tracking systems. In the Jira, which is the issue tracking system of the Qt project, the commit hashes are reported at *changelog.history* field. In contrast, in the Bugzilla issue tracking system, commit hashes are reported at *see_also* field for LibreOffice and Eclipse. For Wireshark, instead of commit hashes of linked PRs, we have the URLs of linked PRs in the issue data. Using a slightly different

algorithm, we use PR URLs instead to retrieve commit hashes of these linked PRs (line 4 of Algorithm 2).

After fetching a list of commit hashes reported in issues, Algorithm 3 compares commit hashes reported in the issue data with the ones in the review data, which are reported in the *revision* field of each PR in Gerrit projects. In comparing reported commits in review data and issue data, if a link is not found, a missing linkage occurs. Also, if the description in the commit message is less than two lines (excluding the title and footer lines), a short description occurs. If PR experiences missing linkage or short description, missing context smell occurs.

Algorithm 3 Missing linkage between commits and issues

```
1: for each PR in review history do
2:   commit_message  $\leftarrow$  PR.CommitMessage
3:   subject  $\leftarrow$  PR.Subject
4:   lines  $\leftarrow$  commit_message.splitlines
5:   short_description  $\leftarrow$  False
6:   issue_ID_in_message  $\leftarrow$  False
7:   common_commits_in_issues  $\leftarrow$  False
8:   Line_start  $\leftarrow$  ("Change-Id:", "Reviewed:", "Signed-off-by:")
9:   if lines < 2 then
10:    short_description  $\leftarrow$  True
11:   if lines = 2 and lines[0] = subject then
12:     if lines[1].startswith Line_start then
13:       short_description  $\leftarrow$  True
14:   if issue_ID in commit_message then
15:     issue_ID_in_message  $\leftarrow$  True
16:   if any change_requestCommits in commits_in_issueData then
17:     common_commits_in_issues  $\leftarrow$  True
18:   if not short_description then
19:     return False
20:   if issue_ID_in_message then
21:     return False
22:   if common_commits_in_issues then
23:     return False
24:   return True
```

Root causes of this smell are (1) negligence of the author to report related issue's ID and in the PR's commit message and (2) negligence of the PR's author to write a proper description to explain the aims of the changeset.

To detect this smell, the following steps are performed:

1. Only PRs with "merged" and "abandoned" status are considered.

2. Commit hashes reported in the issue data are fetched.
3. Commit hashes of each PR are compared to the ones reported in the issue data.
4. Commit message of each PR is checked for reported issue IDs.
5. If (1) none of the commit hashes of a PR are linked to an issue and (2) no issue ID is reported in the commit message of the PR, the PR is considered to have a missing linkage.
6. If the number of lines (written by the author of the PR) in the commit message, excluding the title and the Gerrit-specific footer lines such as ‘*Change-Id*’ or ‘*Reviewed-by*’, is less than two lines, we mark it as a PR with a short description.
7. If missing linkage or short description are observed in a PR, the PR is considered to have the missing context smell.

3.3 Review Buddies Smell

Review buddies smell refers to the situation where the PR author tends to have their PRs reviewed by the same reviewer each time. Based on the review buddies smell detection algorithm defined by Dogan and Tuzun [1], if a reviewer has reviewed at least half of the commits submitted by an author, then this reviewer is the review buddy of the author. We improved this algorithm by using a statistically more dynamic threshold. In an improved detection algorithm, if the occurrence frequency of an author-reviewer pair in the whole project is more than a dynamic threshold which is $Third\ quartile + 1.5 * interquartile\ range$ of frequency distribution of all author-reviewer pair, that reviewer is the author’s review buddy. The third quartile (Q3), is the value under which 75% of data points are located when arranged in ascending order. In addition, The interquartile range provides information about the spread of a distribution’s middle half. It indicates where the ”middle 50%” is located. This threshold is the maximum

value of the frequency distribution without considering the outliers while visualizing the frequency of all pairs with a box plot. We set the threshold to this value since when the occurrence frequency of a pair is not in the [minimum value, maximum value] range of the box plot of frequency without considering outliers, the pair is over-occurred, and the reviewer has become author's buddy. This scenario eliminates non-merged, self-reviewed, non-reviewed, and bot-reviewed commits. To investigate this smell, the following steps are performed:

1. For each PR, we construct author-reviewer pairs. For example, for a PR with author (A) and n reviewers ($R_1, R_2 \dots R_n$), we construct n pairs as ($AR_1, AR_2 \dots AR_n$). In these pairs, the order of the developers matters. For example, the pairs (AB) and (BA) are different.
2. The occurrence frequency of each author-reviewer pair which refers to the number of occurrences of a specific pair in all review history of a project, is calculated.
3. To calculate a threshold to separate review buddies and non-review buddies based on the occurrence frequency of each pair, the third quartile (Q3) and interquartile range (IQR) of all pairs' occurrence frequencies are used to establish a threshold as seen below. High frequent pairs are considered review buddies while low frequent pairs are normal, standard pairs.

$$\mathbf{Pair} = \begin{cases} \text{Low frequent pair} & \text{if occurrence frequency} < Q3 + 1.5 * IQR \\ \text{High frequent pairs} & \text{otherwise} \end{cases}$$

Low frequent pairs := { pairs | occurrence frequency < $Q3 + 1.5 * IQR$ }

High frequent pairs := { pairs | occurrence frequency > $Q3 + 1.5 * IQR$ }

3.4 Categorizing Code Review Process Smells

In this section, we categorize code review process smells based on whether the problem is on the reviewers' or the committers' side and then illustrate each smell's start point in the lifecycle or a PR. Table 3.1 shows that each smell is reviewers' side or committers' side and also reports the start point of each smell in the lifecycle of a PR. In Table 3.1, smells illustrated with one star, "*", are the ones that their detection algorithms are modified and improved compared to Dogan and Tuzun's smell detection algorithm [1]. Smell with two star, "**", is defined, detected, and analyzed by us. Other smells (without a star) are detected based on the Dogan and Tuzun's original detection algorithm [1]. Figure 3.2 also illustrates the initial creation point of each smell in a PR. Smells initiated at PR submitting or assigning reviewer phases are committers' side smells. On the other hand, smells that are initiated during the review process are reviewers' side smells. Moreover, related to bypassed negative reviewer smell, even though bypassed negative reviewer smell is initiated when the pull request is merged with an unchanged negative vote, the potential of this smell is created when a new reviewer is assigned (positive reviewer) after the negative reviewer's vote, so technically, the potential of this smell is created at the phase of assigning reviewers.

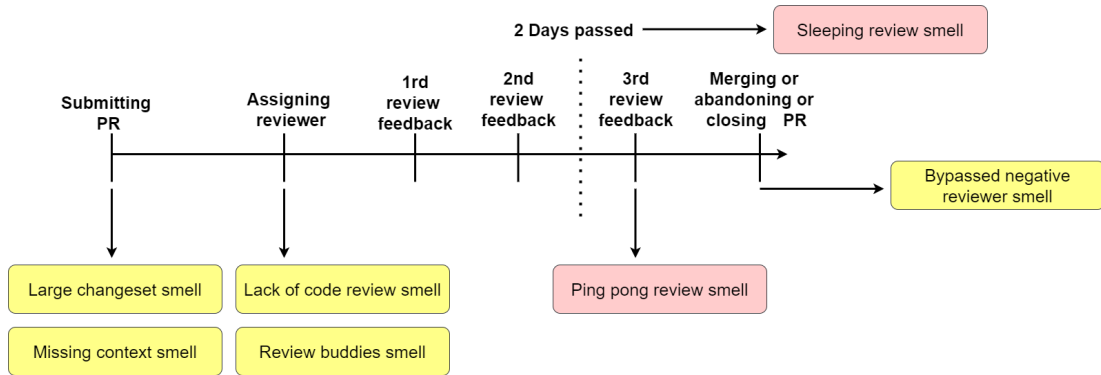


Figure 3.2: Pull request lifecycle and start point of each code review process smell

Table 3.1: Categorizing code review process smells

Smells	Initiator	Start point
Lack of code review	Committer	Assigning reviewer
Ping pong review	Reviewer	After 3 iterations
Sleeping review	Reviewer	After 2 days
Large changeset	Committer	Submitting PR
Missing context*	Committer	Submitting PR
Review buddies*	Committer	Assigning reviewer
Bypassed negative reviewer**	Committer	Merging PR (with negative vote)

3.5 Correlations Between Code Review Process Smells

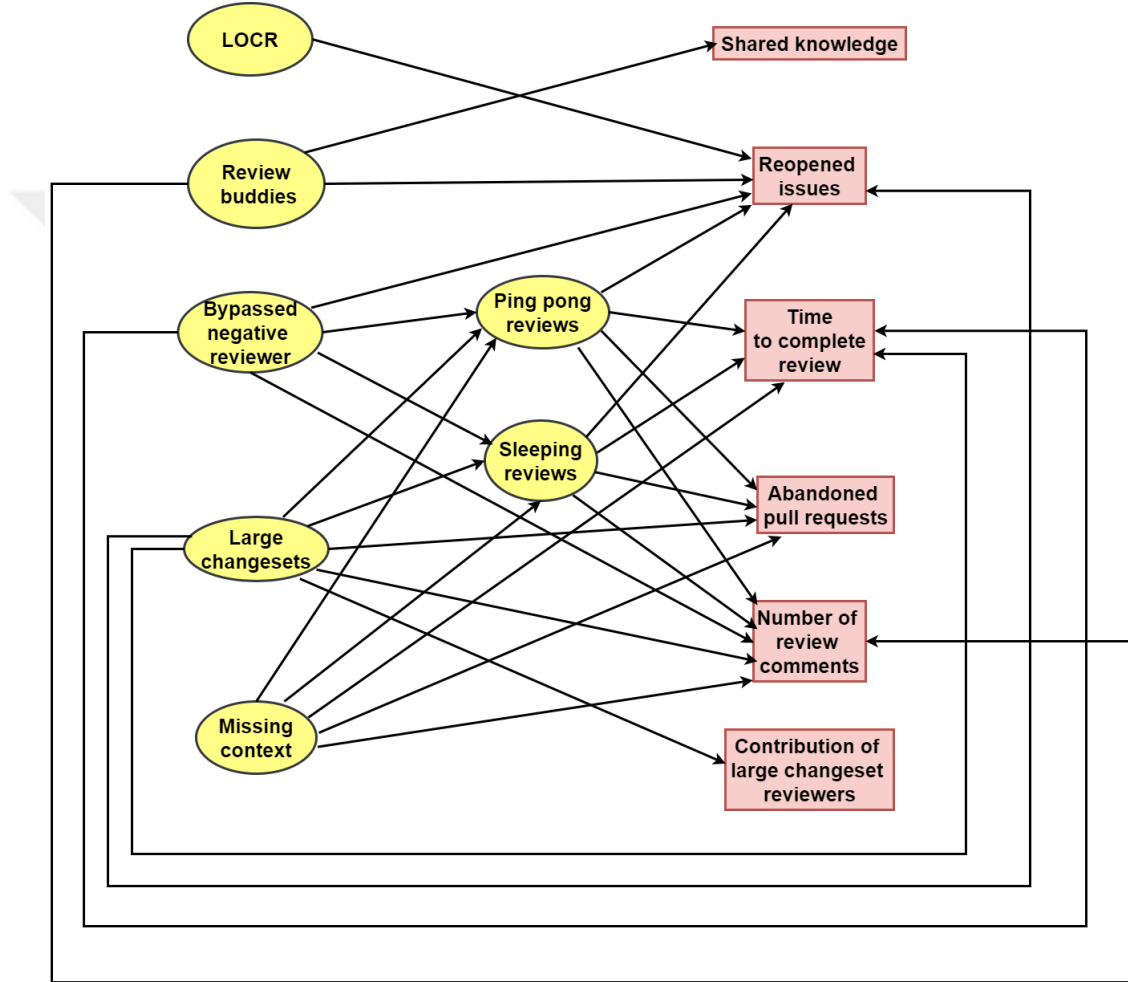


Figure 3.3: Correlations between code review process smells and their impacts

In the previous section, we categorized code review process smells into committers' side smells, and reviewers' side smells. The existence of committers' side smells in a PR may lead to the occurrence of reviewers' side smells in the same PR in the code review process as nonconformance of the committer to the formal software development practices may lead to reviewers' bad practices. Note that even though lack of code review is a committers' side smell, since there is no

reviewer other than the author, the correlation between this smell and reviewers' side smells cannot be analyzed. Figure 3.3 shows a graph depicting smells (in circles) and correlations between these two categories as well as impacts of code review process smells (in rectangles). In Figure 3.3, we model potential correlations among code review process smells and their impacts. For instance, the bypassed negative review smell in a PR may lead to ping pong, and sleeping review smells in the same PR. Some of the impacts exhibited in Figure 3.3 are specific to one of the smells, such as shared knowledge impact or contribution of large changeset reviewers impact. However, for other impacts to specify which smell is the potential reason, we considered the definition and characteristics of each smell and impact. For instance, missing context smell cannot lead to reopened issues since PRs with this smell suffer from missing linkage to an associated issue, so we cannot measure linked issues' reopen ratio in this case. Moreover, the time to complete review impact cannot be measured for lack of code review smell. Furthermore, the abandoned pull requests impact is only measurable for smells that PRs with abandoned status are considered in the definition and detection of these smells.

Note that the LGTM smell cannot be detected easily because PRs with LGTM smell may be PRs without any defect, and no feedback might be required in the code review of these PRs. The source code of the submitted commits should also be analyzed to detect the LGTM code review process. Since we are analyzing the code review process smells by mining the code review data of projects and the source code of commits are not included, LGTM smell cannot be detected. Furthermore, this smell was not quantitatively analyzed by Dogan and Tuzun [1] which is why we discard this smell from the graph in Figure 3.3.

3.6 Impact Of Code Review Process Smells

Code review process smells arise when developers deviate from the ideal code review process, and there are consequences to introducing these smells. We illustrate these consequences by investigating the impacts of smells in the code

review process. We categorize six impacts (shared knowledge, reopened issues, time to complete review, abandoned pull requests, number of review comments, and contribution of large changeset reviewers) illustrated as rectangles in Figure 3.3. In the following, we will go over the six different types of impact:

3.6.1 Shared knowledge impact

One of the benefits of code review is balanced knowledge and codebase ownership within the development team which can be achieved by developers reviewing different PRs related to different parts of the codebase. Bad practices in the code review process can reduce the balance of shared knowledge of the development team and may put most of the review workload and burden on specific members of the development team and create inequality. This impact is very similar to the social inequality concept in sociology that occurs when resources in a given society are distributed unevenly. While dealing with this impact, inequality happens when the review workload is distributed unevenly. In the code review process, smells such as review buddies can lead to this problem. To demonstrate this impact, we assign each reviewer a codebase coverage score (**CBCS**) that indicates the sum of reviewed lines of code (LOC) by each particular reviewer, reviewed for PRs' of an author that the reviewer is paired with. This score indicates the amount of work a reviewer has done for an author if they belong to the same pair. We compare the workload (CBCS) of the review buddy and non-review buddy pairs. Reviewed lines of code are computed as the sum of the number of inserted and deleted lines in the final (last) commit for Gerrit projects. In Gerrit projects, each code review operation has a change kind label that match if the difference between the changes made on a file (diff) is of a specific change kind; "Rework", "Trivial rebase", "No code change", and "No change". Each of these change kinds refer to the purpose of code review. The kind that corresponds to the work done to address a reviewer's feedback is named "Rework" and refers to nontrivial content changes in the codebase. "Trivial rebase" label refers to a situation when the commit message is the same as in the previous changeset and if it has the same diff (including context lines) as the previous changeset. "No change" label

refers to a situation when the parent tree, code delta, and commit message do not change from the previous changeset, whereas the "No code change" label means that the new changeset has the same parent tree and the same code delta with the previous changeset [20]. In the calculation of reviewed lines of codes in Gerrit projects, only "insertions" and "deletions" with "Rework" kind in the last commit of a pull request are considered. For instance, if in author-reviewer pair of **A-B**, **B** is the review buddy of **A** as this pair is observed n times in the code review history of the project which $n > Q3 + 1.5 * IQR$ (the dynamic threshold used in the definition of review buddies smell), thus reviewer **B**, reviewed n pull requests of author **A**, and the CBCS score of reviewer **B** is the sum of "Rework" kind inserted and deleted lines of code in the all mergeable commit(s) in all of these n pull requests which reviewer **B** have reviewed.

Figure 3.4: Scheme of shared knowledge impact detection algorithm in review buddies smell

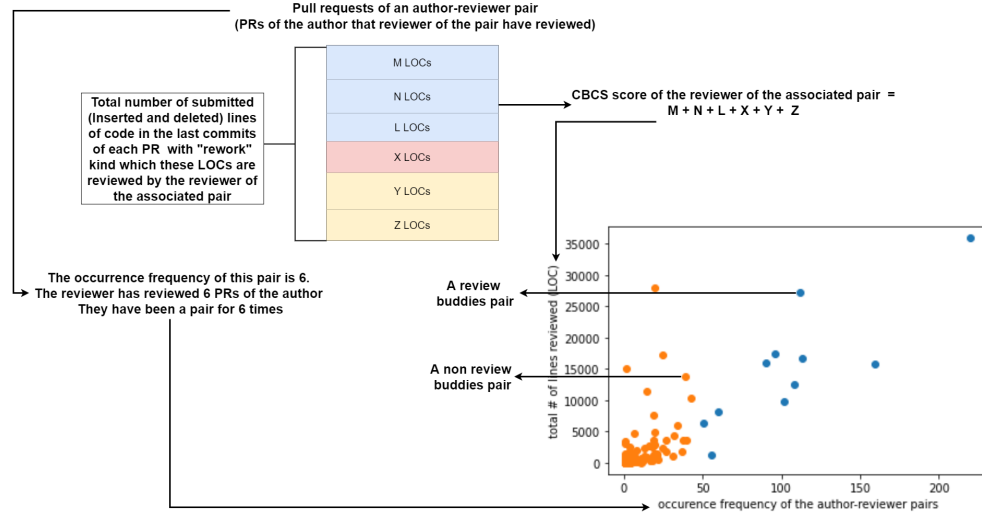


Figure 3.4 shows the algorithm of shared knowledge impact in review buddies smell, how each reviewer's CBCS score is calculated in a pair, and how we construct a plot in which each dot represents a pair, and by this, the occurrence frequency of a pair and CBCS score of its reviewer are comparable. By calculating the workload of high frequent pairs (review buddies) and low frequent pairs (non-review buddies), we can measure the unbalancedness of shared knowledge

in the whole project in the existence of review buddies smell.

3.6.2 Reopened issues impact

Ideally, issue reports and PRs should be linked. In the code review process, every commit in a PR may be linked to an associated issue in the issue tracking system that can be reopened after it is closed if the reported issue is not resolved correctly. Our chosen projects use Jira or Bugzilla as their issue tracking tool in our study.

Jira and Bugzilla adopt different issue workflows. Furthermore, Jira and Bugzilla allow having custom issue statuses on top of the default ones. Figure 3.5 shows the lifecycle of an issue in Jira. In Jira, an issue is “reported” by a user, then its status is changed to “open”, “need more info”, “in progress”, “resolved”, “verified”, “in progress”, “implemented”, and “closed”. In Jira, an issue reopens when its status changes in the direction of red arrows in Figure 3.5. If an issue’s status history includes the status change in the direction of red arrows in Figure 3.5, we consider that issue as a reopened issue, since the issue has a reopened transition in its status history. In Bugzilla, an issue’s status can change to “new” after it is reported and then “unconfirmed” or “confirmed” or “assigned”, and then “in progress”, “resolved”, “verified”, “closed” and “reopen”. An issue can be reopened in Bugzilla for the same reasons as well. Figure 3.6 shows the lifecycle of an issue in the Bugzilla issue tracking system. In order to investigate the reopen impact, we gathered all the statuses found in the issue data for the four projects we had and then categorized them into *Closed*, *Open*, and *Reopened* categories. In Jira, reopened issues are the ones whose history of status change contains any transition from any *Closed* category (“Closed”, “Resolved”, “Implemented”, “Verified”) to any *Open* category (“Open”, “In Progress”).

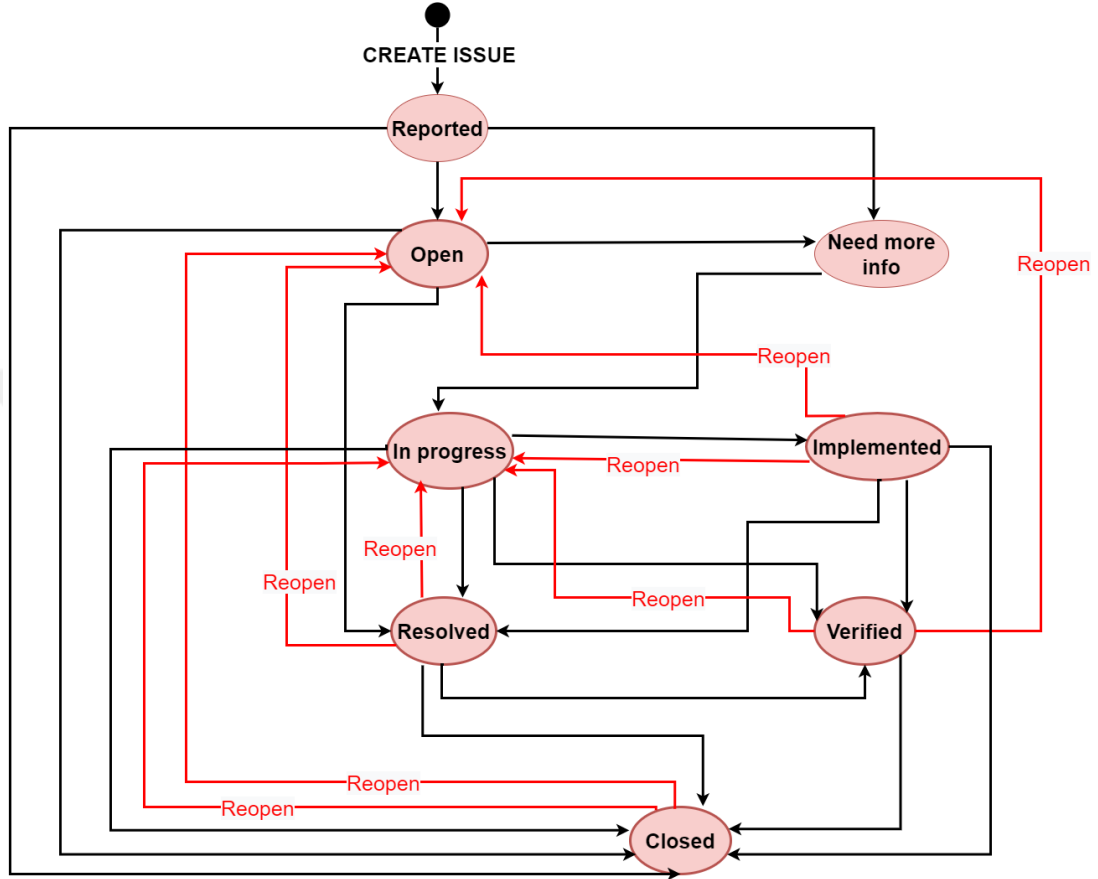
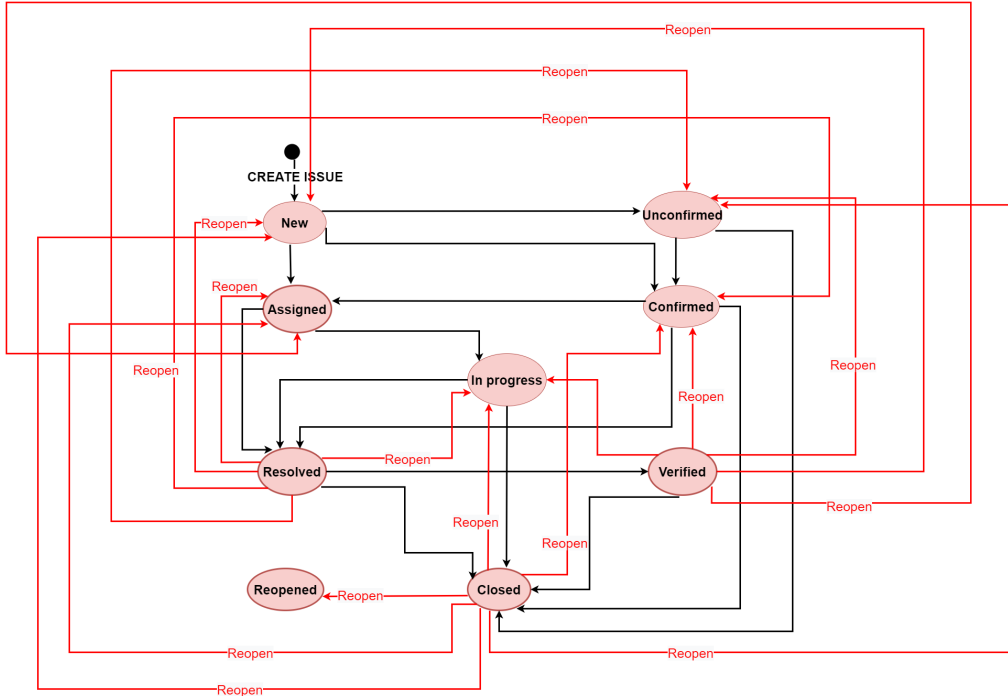


Figure 3.5: Lifecycle of an issue in QT project issue tracking tool (Jira)

Bugzilla also similarly stores status change history. Here we consider any issue which has a status change from any *Closed* category (“resolved”, “verified”, “closed”) to any *Open* category (“confirmed”, “in progress”, “new”, “assigned”, “unconfirmed”) as well any change to a *Reopened* status (“reopened”) as a reopened issue. This ensures that regardless of the current status if an issue was reopened at any point in its history, it is counted as a reopened issue. After detecting reopened issues, both issue numbers and linked commit hashes reported in reopened issues are retrieved to detect pull requests linked to reopened issues. If a commit message contains information about the fix of an issue with issue ID, which is considered a reopened issue, then the pull request to which the commit belongs is considered a pull request linked to a reopened issue. If a commit hash of a PR is reported in a reopened issue, the pull request to which the commit

belongs is considered a pull request linked to a reopened issue.

Figure 3.6: Lifecycle of an issue in Eclipse, LibreOffice and Wireshark projects issue tracking tool (Bugzilla)



Moreover, reopen ratio refers to the percentage of PRs linked to reopened issues among all PRs linked to any issue (PRs with linkage to an associated issue). Bad practices in the code review process can increase the issue reopen ratio in projects. All smell types, except missing context smell, may likely get the smelly PR's linked issue reopened later because of the potential problems in the pull requests. We excluded missing context smell because, based on the definition of this smell, the pull request is not linked to an issue, so we cannot track the issue.

3.6.3 Time to complete review impact

The key motivation of the code review process is detecting potential problems as early as possible in the development cycle to save time, effort, and resources, and improve source code quality [21]. This practice should be completed in a reasonably timely manner. The time to complete review is the duration from when the first changeset is submitted until the last human activity (comment, status change, and similar other actions) is performed for the PR (actions performed by bots are discarded). Reviews that take too long to be completed negatively affect the software development process and slow down, disrupt, or even block the development workflow. Smells such as (1) sleeping review, (2) bypassing the negative reviewer, (3) reviewer-author ping pong, (4) large changesets, and (5) missing context smells can extend the time to complete review compared to non-smelly PRs. To investigate this type of impact, the following steps are performed:

1. All comments and operation messages made by recognized bots are filtered out for each PR so that only those originating from humans remain (bot detection procedure is discussed in Section 4.2). From these remaining items, the completion time of each PR is calculated by subtracting the timestamps of the first submitted patchset and the last non-bot comment. Then calculated time is converted from seconds to days.
2. Time to complete review of smelly PRs and non-smelly (normal) PRs are calculated, and distribution of TTCR of smelly and non-smelly PRs is compared. To compare the distribution of TTCR in different categories, the box plot of the values in smelly PRs and non-smelly PRs is used.

3.6.4 Abandoned pull requests impact

Sometimes developers, as authors of changesets in the PRs, face challenges getting their PRs approved. These challenges can cause the author to abandon the PR. Abandoning PRs cause the problem that changeset is committed to solve to

remain unresolved. Even if the author abandons the PR and opens a new PR, the associated problem has an abandoned PR that remains unresolved. Smells such as (1) reviewer-author ping pong, (2) sleeping reviews, (3) large changesets, and (4) missing context smells may cause abandoned PRs impact. Numerous iterations of reviewer-author ping pong smell, long time to complete review in sleeping reviews, difficulty of reviewing large changesets and PRs with missing context smell might exhaust the author and cause the PR to be abandoned. To investigate this type of impact, the following steps are performed:

1. Ping pong, sleeping, large changeset, and missing context smells are detected.
2. The ratio of PRs with "abandoned" status among these smelly PRs is calculated.

3.6.5 Number of review comments impact

One of the crucial steps of an ideal code review process is to give feedback on the changesets if there are potential problems to the author to make modifications, fix the possible problems, and improve the source code. In the Gerrit projects, two types of comments are observed: (1) inline comments and (2) file-level comments. Inline comments are given directly in the code changes and code insertions. File-level comments are added to the changed files and do not deal with the code itself. Some developers perform code reviews without giving feedback and comments on the code changes and directly approve the changeset. In this situation, code changes may get merged into the codebase with some potential problems. To investigate the differences between the number of review comments in smelly and non-smelly PRs, the sum of inline comments and file-level comments given by the reviewers of a pull request was computed. Smells such as (1) review buddies, (2) bypassed negative reviewer, (3) large changeset, and (4) missing context smells can lead to different distributions of the number of review comments in smelly and non-smelly PRs. To investigate this impact, the following steps are performed:

1. Sum of inline and file-level comments given by reviewers on the code insertions and deletions in each pull request of Gerrit projects are calculated.
2. Distributions of the number of review comments in two different categories (smelly and non-smelly PRs) are compared. To compare distributions, box plots are used, and to justify the significance of differences statistical tests are performed.

3.6.6 Contribution of large changeset reviewers impact

By reviewing a large changeset, a reviewer's number of reviewed lines of code and received knowledge increase dramatically. This received knowledge can affect a reviewer's future contribution to the project.

Knowledge can be classified into two categories; explicit and tacit. Explicit knowledge is already codified, whereas tacit knowledge is rooted in mind. We expect that a large changeset reviewer by gaining knowledge, may contribute more to upcoming PR reviews. Recognizing the pattern of reviewers in reviewing large changesets in the code review history of projects and investigating how balanced this pattern is, enable us to measure the contribution of large changeset reviewers in projects. On the other hand, if the review of a PR with large changesets is not performed properly and attentively, increase in the gained knowledge by associated reviewer may create a bias. This means that even though a reviewer received knowledge by reviewing a PR with large changesets if the review task is not applied properly with precision, this knowledge is not tacit but only explicit. This explicit knowledge is the created bias that may cause the large changeset reviewers to be considered qualified more than how much they are experts in code review or how much they have code/file ownership. To investigate this impact, the following steps are performed;

1. We sort the data according to the creation time of pull requests. Then we divide the data into ten equal-sized subsets.

2. Each pull request is categorized into one of the size categories; small, medium, and large changesets.

$$\text{Size category} = \begin{cases} \text{Small} & \text{if } \text{Changed LOCs} < 50 \\ \text{Medium} & \text{if } 50 < \text{Changed LOCs} < 200 \\ \text{Large} & \text{if } \text{Changed LOCs} > 200 \end{cases}$$

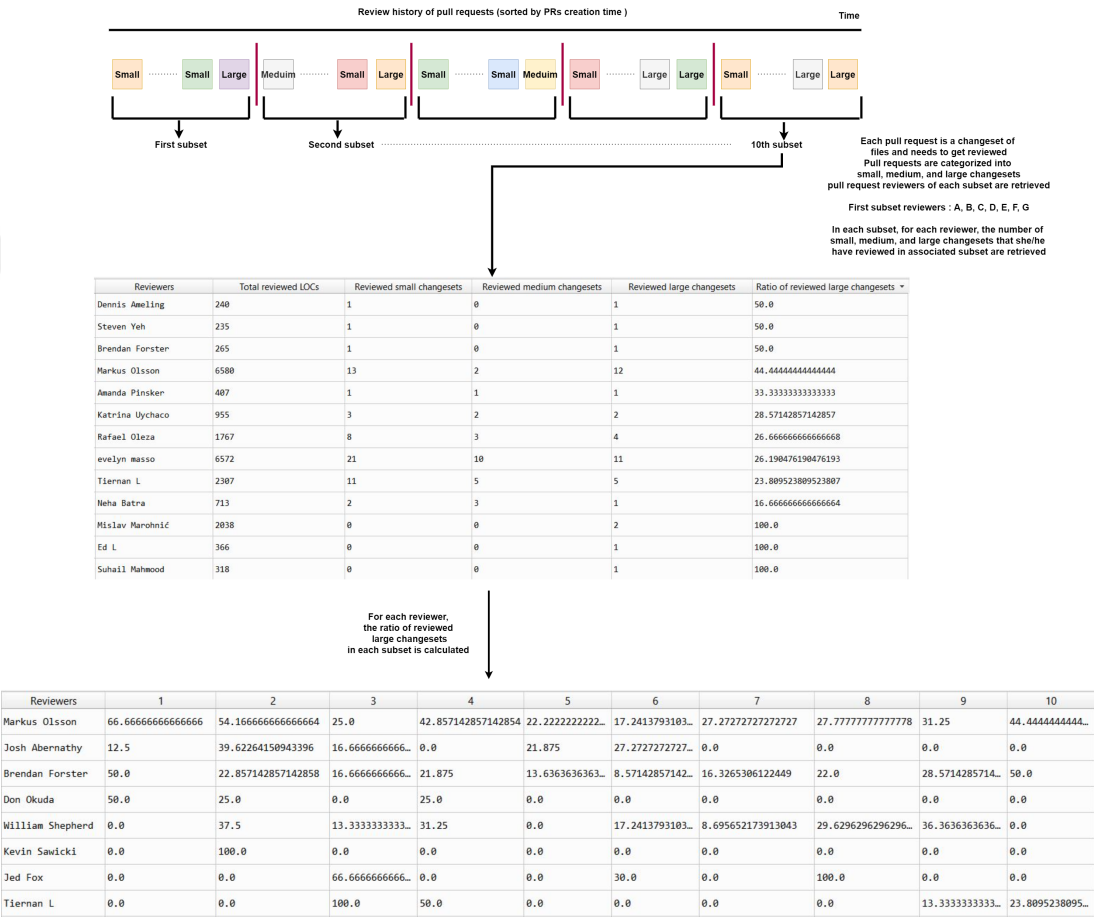
3. In each subset, we retrieve the reviewers and the number of PRs with small, medium, and large categories that each reviewer has reviewed.
4. For each reviewer in a subset, we calculate the ratio of reviewed large PRs in associated subset as :

$$\text{Ratio of reviewed large PRs} = \frac{\text{Number of reviewed large PRs}}{\text{Total number of reviewed PRs}}$$

5. The ratio of reviewed large PRs of reviewers in each of 10 subsets are calculated
6. We compare the trend of change in the contribution of reviewers during time by measuring and comparing the ratio of reviewed large PRs in each subset.

Figure 3.7 shows the algorithm's scheme to measure the reviewers' contribution in different subsets. In Figure 3.7, the first table shows the contribution of reviewers in one of the subsets (10th subset), the total number of reviewed LOCs, and the number of reviewed small, medium, and large PRs. The second table illustrates the ratio of reviewed large PRs for each reviewer in each subset. We use this ratio as an indicator of reviewers' contribution over time. By analyzing the pattern of changes in the value of this ratio for each reviewer in each subset, we can observe the contribution pattern of reviewers in projects.

Figure 3.7: Scheme of algorithm to detect the pattern of contribution of large changeset reviewers



Chapter 4

Results

This chapter will provide some details about projects, including the total number of pull requests in the code review history of chosen projects and the total number of issues in issue tracking data in Section 4.1. Afterward, we will continue with the preprocessing steps we take before our analysis in Section 4.2. Next, we will provide our results in correlations between smells in Section 4.3 and results related to impacts of code review process smells in Section 4.4.

4.1 Code Review and Issue Tracking Datasets

To investigate the real-life impacts of smells in the code review process, we mined four open-source projects which use Gerrit as their code review tool: Qt, Eclipse, Wireshark, and LibreOffice. Gerrit is a recent code review tool that is free and web-based. In Gerrit, authors’ subsequent changes to the initially submitted patchset in response to reviewer feedback appear in the same review and are referred to as “revisions,” or “changesets” [22].

In the data collection process, we used the Perceval tool [23] from GrimoireLabs to retrieve datasets from Gerrit, Jira, and Bugzilla. The summary statistics for code review data of chosen four projects are given in Tables 4.1 (filtered reviews

are the reviews of PRs with only "merged" status). Furthermore, details of issue tracking data of projects are given in Tables 4.2 (reopened issues are the ones that had a status transition from *Closed* states to *Open* states, or their status directly changed to reopened in their status change history in the issue tracking data).

Table 4.1: Details regarding the code review data of four Gerrit projects used

Project name	Total reviews	Filtered reviews	Starting date	Ending date
Qt	96,722	80,103	2017-01-01	2020-04-20
Eclipse	71,993	58,656	2017-01-01	2020-04-20
Wireshark	17,407	16,373	2017-01-01	2020-04-20
LibreOffice	58,781	54,348	2017-01-01	2020-04-20

Table 4.2: Details regarding the issue tracking data of four Gerrit projects used.

Project name/ Issue tracking system	Total issues	Reopened issues
Qt/Jira	25,019	1,029
Eclipse/Bugzilla	45,923	2,393
Wireshark/Bugzilla	3,162	74
LibreOffice/Bugzilla	24,800	1,583

4.2 Data Preprocessing

After fetching the data by Perceval [23], an inspection of the preliminary and unprocessed data is performed for the projects discussed above.

1. In Gerrit projects, PRs with the "new" status were not considered in the analysis for any metric. Only "merged" and "abandoned" PRs were analyzed.
2. Only "merged" PRs were considered for analyzing lack of code review, review buddies, and bypassed negative reviewer smells because based on the

definition of these smells, PRs should get merged at the end of the lifecycle of the PR. For these smells, “abandoned” reviews were not considered.

3. Our inspection of each developer in the projects revealed that projects contain non-human agents such as review bots. For example, the most common reviewer name in the Eclipse project is CI Bot which appears in 37.4% of the PR reviews. We extracted all the names in the reviewer name and author name fields in the data for all four projects and created a list of contributors for each project. These lists were manually inspected to find all non-human names which are bots. This list was then used to filter out non-human reviewers and authors.
4. Each PR in Gerrit may have several “revisions” or “patch sets”. Each revision is equivalent to an iteration of a commit, showing modification consisting of deletions and insertions done by the author. According to the convention that Gerrit adheres to, it is only the most recent revision uploaded to a PR that is applied to the target branch. We picked the most recent revision for each PR to calculate the total number of changed LOCs (inserted and deleted lines). We considered traversing all revisions for a given PR and summing up the changed LOCs for all of them, but there were cases where there was a significant difference in the content of previous revisions of a PR and the most recent one that Gerrit picks and applies. In one case, a previous revision had more than 3 million lines changed while the applied (the most recent one) revision merely had a few hundred changed lines.
5. Smell and impact detection for individual smells were skipped where the PRs were missing the relevant information. For example, a negative review in a PR was skipped if it was missing reviewer ID information while running the bypassed negative review algorithm.

4.3 Correlation Results

The existence of committers' side smells may lead to the occurrence of reviewers' side smells in the code review process as the bad practice of committer in submitting PR phase or assigning reviewer phase might disadjust reviewers' behavior. Related to co-occurrence of these smells, there are two cases; positive case and negative case. Co-occurrence of a committers' side smell and a reviewers' side smell simultaneously in the same PR (double-positive), or a PR without any of these particular smells (double negative) is called a positive case. Occurrence of a committers' side smell in a PR without the existence of reviewers' side smell (single positive, single negative), or occurrence of one of a reviewers' side smell in a PR without the existence of committers' side smell (single negative, single positive) is called negative case. When the occurrence of a particular reviewers' side smell is likely by the existence of associated committers' side smell in a PR (PR with both smells. double-positive), and also occurrence of the reviewers' side smell is not very likely when the committers' side smell does not exist in the PR (PR without both smells, double-negative), a correlation between these smells exist. The correlation percentage is the percentage of PRs that already have one type of committers' side smell and exhibit a reviewers' side smell in the same PR as well. In the following, we will report the correlation ratios between different smells.

4.3.1 Bypassed negative reviewer smell vs. sleeping review smell and ping pong smell

The first correlation we analyzed through all four projects is between the bypassed negative reviewer and the sleeping review smell. Negative reviewer and reviewer reassignment for a positive vote in bypassed smell may cause the PR to be reviewed in more than two days and create a sleeping review smell. Another correlation that we investigated is between bypassed negative reviewer smell and ping pong smell as to bypass the negative reviewer, another reviewer (positive

Table 4.3: Bypassed negative reviewer smell frequency in four Gerrit projects

Projects	Merged PRs	Bypassed negative reviewer smell frequency	Smell Perc.(%)
Qt	80,103	142	0.17
Eclipse	58,656	114	0.19
Wireshark	16,373	47	0.28
LibreOffice	54,348	58	0.10

Table 4.4: Correlations between bypassed negative reviewer smell vs sleeping review smell and ping pong smell and correlation percentage (Corr. perc.%) in four Gerrit projects

Projects	Sleeping reviews among bypassed reviews	Corr. perc.%	Ping pong reviews among bypassed reviews	Corr. Perc. (%)
Qt	97	68.30	120	84.50
Eclipse	66	57.89	22	19.29
Wireshark	23	48.93	38	80.85
LibreOffice	31	53.44	9	15.51

reviewer) needs to be added, which leads to more iterations.

Table 4.3 shows the frequency of bypassed negative reviewer smell in four Gerrit projects among merged PRs, and Table 4.4 shows the percentage of sleeping reviews among PRs with bypassed negative reviewer smell. In all four projects, 48-68% of the bypassed reviews take more than two days to be reviewed and merged and suffer from sleeping review smell. Related to the correlation between bypassed negative reviewer smell and sleeping review smell, the double-negative (positive) case is also very likely; however negative cases are not very probable.

Table 4.4 also illustrates the ratio of ping pong reviews among bypassed reviews. Table 4.4 reveals that more than 80% of the bypassed reviews contain more than three review cycles and suffer from ping pong smell in Qt and Wireshark projects. On the other hand, this ratio is relatively low in LibreOffice and Eclipse projects (15.51% and 19.29%, respectively). One of the reasons that this percentage is low in the LibreOffice and Eclipse projects is that ping pong smell is not very likely to occur in Eclipse and LibreOffice projects and the overall

Table 4.5: Correlations between large changeset smell (LCHS) vs sleeping review smell and ping pong smell in four Gerrit projects

Projects	LCHS Smell Freq.	Sleeping reviews among PRs with LCHS	Corr Perc. %	Ping pong re-views among PRs with LCHS	Corr Perc. (%)
Qt	9,893	6,953	70.28	4,434	44.81
Eclipse	13,845	6,013	43.43	1,301	9.39
Wireshark	2,285	585	25.60	311	13.6
LibreOffice	6,345	1,864	29.37	127	2.0

ratio of ping pong smell in these projects is very low (2.3% and 5.1% respectively). In addition, these projects are highly likely to observe double negative (positive) cases. Furthermore, the LibreOffice project has the lowest bypassed negative reviewer smell ratio. One of the reasons is that the LibreOffice contribution guideline explicitly warns developers that a PR cannot be merged as long as there are negative votes. Despite this warning, we observed 58 PRs with bypassed negative reviewer smell in the LibreOffice project.

4.3.2 Large changeset smell (LCHS) vs sleeping review smell and ping pong smell

We expected that since reviewing PRs with unfeasibly large changesets with a great number of code changes (insertions and deletions) is difficult; this may make the review process too long to complete and lead to sleeping review smell. Another correlation that was expected is between large changeset and ping pong smells. Since the full, proper, and attentive review of PRs with large changesets requires more effort, feedback, and modifications, ping pong smell may occur in the PRs with large changesets. However, since reviewing PRs with large changesets is a difficult task for reviewers, reviewers may not want to deal with a proper code review of such PRs, and the ratio of sleeping and ping pong smells may be less than our expectation.

Table 4.6: The average number of iterations in the review and feedback cycle of PRs in different changeset (CHS) size categories in four Gerrit projects

Projects	Average #of iterations in large CHS	Average #of iterations in medium CHS	Average #of iterations in small CHS
Qt	4.6	3.3	2.5
Eclipse	1.34	1.07	0.73
Wireshark	1.48	1.45	1.19
LibreOffice	0.57	0.67	0.81

As it is illustrated in Table 4.5, in the Qt project, the ratio of sleeping reviews among large changesets is the highest (70.28%) with a significant difference since the contribution guideline of this project explicitly mentions that "give reviewers ample time to respond" [24]. This ratio is 43.43% in the Eclipse project. In addition, in the Wireshark and LibreOffice projects, even though 25.60% and 29.37% of the PRs with large changesets took more than two days to be reviewed, respectively, which is not a very high ratio, the likelihood of observing double-negative (positive) case is also high; hence it is very likely that if a PR does not consist of the large changeset, sleeping review smell won't occur. Table 4.5 also shows the ratio of ping pong reviews among PRs with large changeset size as in all projects except Qt project, less than 14% of PRs with large changesets have ping pong smell. The ratio of ping pong smell among PRs with large changesets of Qt project is 44.81%. Qt project behaves differently in terms of sleeping and ping pong smells since the ratio of sleeping and ping pong smells are high in Qt project, and it is widespread for a PR that its review takes more than two days and more than three iterations to be reviewed. Even though the ratio of ping pong smell among PRs with large changesets is less than 14% in Eclipse and Wireshark projects, the likelihood of observing negative cases is less than positive cases. This means that if large changeset smell does not exist in a PR, occurrence of ping pong smell is not very likely. On the other hand, negative cases are very likely in LibreOffice project. Moreover, Table 4.6 displays the average number of feedback and modification iterations in three different categories; large (more than 200 LOCs), medium (50-200 LOCs), and small (less than 50 LOCs)

Table 4.7: Missing context smell frequency in four Gerrit projects

Projects	Merged and abandoned PRs	Missing Context Smell Freq.	Smell Perc.(%)
Qt	92,239	22,260	24.13
Eclipse	69,686	24,439	35.07
Wireshark	17,287	5,170	29.90
LibreOffice	58,491	24,088	41.18

changesets as in all projects except LibreOffice, the average number of iterations in PRs increases with the size of changesets. One reason that we are not able to observe the direct relationship between the size of changesets and the average number of iterations in the LibreOffice project is that the ratio of ping pong smell is the lowest in the LibreOffice project (2.0% among PRs with large changesets) and these two smells are not correlated in LibreOffice project.

4.3.3 Missing context smell vs. sleeping review smell and ping pong smell

We expected that the missing context smell might lead to the ping pong smell. Since missing context PRs suffer from missing linkage or lack of proper description, the reviewer of a missing context PR may need several author-reviewer iterations to understand the purpose of the code changes and the potential issue that the submitted PR is trying to resolve. Another expected is a correlation between missing context PRs and sleeping reviews. Due to missing linkage or lack of proper description in these PRs, reviewing these PRs may require a longer time, and sleeping reviews may occur. Reviewer of a missing context PR may need more time to understand the objectives of the PR and the problem that PR is trying to solve.

Table 4.7 shows the ratio of missing context smell among four Gerrit OSS

Table 4.8: Correlations between missing context smell vs sleeping review smell and ping pong smell in four Gerrit projects

Projects	Sleeping reviews among missing context reviews	Corr. Perc.(%)	Ping pong reviews among missing context reviews	Corr. Perc.(%)	Average #of iterations in Missing Context PRs
Qt	9,130	41.01	4,055	18.21	2.3
Eclipse	4,897	20.03	641	2.62	0.67
Wireshark	600	11.60	431	8.33	1.15
LibreOffice	3,689	15.31	220	0.91	0.55

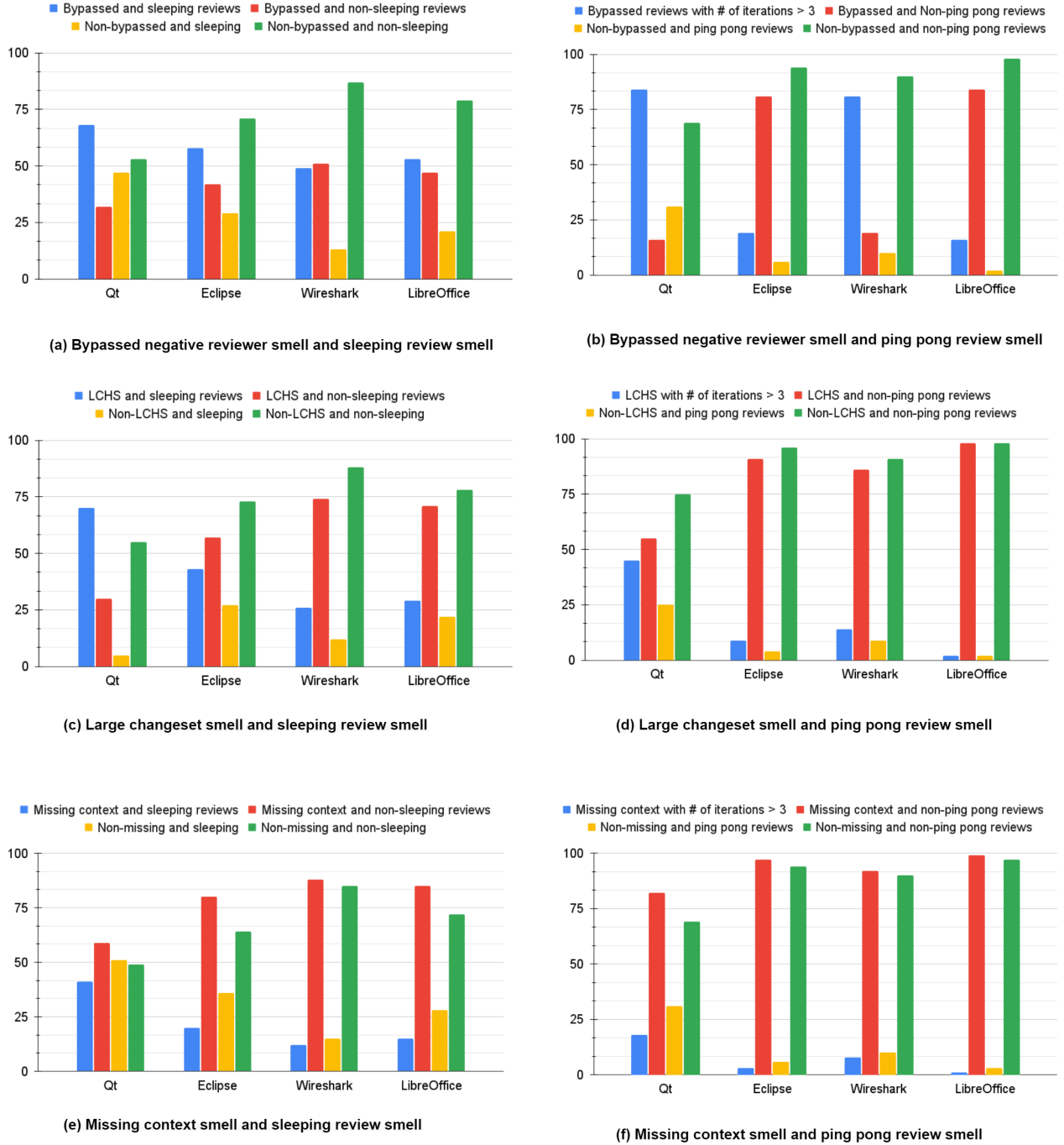
projects which is the highest in the LibreOffice project. Even though contribution guidelines of all projects explicitly mention that missing context and missing objectives should be prevented, the ratio of this smell is between 24-41% in all four projects [24] [25] [26] [27]. Moreover, Table 4.8 shows the frequency of sleeping and ping pong review smells among missing context PRs. 11-41% of missing context PRs suffer from sleeping review smell in all four projects. On the other hand, the ratio of ping pong smell among missing context PRs is very low in all four projects except Qt (0.91-18.33%). In addition, although LibreOffice has the highest ratio of missing context smell, the percentage of ping pong smells among missing context PRs is very low (0.91%). This is because only 2.3% of all the PRs in the LibreOffice project have more than three review iterations and the ratio of ping pong smell is generally low in this project. Despite the ratios illustrated in Table 4.8, we observed that the likelihood of occurrence of positive cases is less than negative cases in both sleeping review and ping pong smells among PRs with missing context smell.

In a nutshell, we analyzed the ratio of sleeping and ping pong smells among PRs with three smells: bypassed negative reviewer, large changesets, and missing context. We observed that the ratio of sleeping smell among these smelly PRs is the highest in Qt projects. This ratio is lower and similar in three other projects. Furthermore, the ratio of ping pong smell among mentioned smelly PRs in the Qt project is very high since sleeping and ping pong smells are pervasive in this project. This ratio is very low in LibreOffice and Eclipse projects since the

general ratio of ping pong smell is meager in these projects. Figure 4.1 shows the percentage of two positive and two negative cases in each project for mentioned smells in the bar charts for co-occurrences and correlations between different cases discussed above. In Figure 4.1, blue and green charts show the positive cases where a smell's existence leads to other smell's occurrence and vice versa, and red and yellow charts show negative cases where only one of the reviewer's side or committers' side smells exist in the PR.



Figure 4.1: Double-positive (blue color), double-negative (green color), [single-positive and single-negative] (red color) ,and [single-negative and single-positive] (yellow color) cases in correlation between different smells



4.4 Impact Results

The quantitative analysis for the impacts of code review smells was performed by applying the impact analysis methodologies that were described in Section 3.6 for four projects.

4.4.1 Shared knowledge impact

To investigate one of the impacts of review buddies, our algorithm analyzed author-reviewer pairs and put a threshold ($Q3 + 1.5 * IQR$) to distinguish high and low frequent pairs as review buddies and non-review buddies respectively.

Figure 4.2: Frequency distribution of author-reviewer pairs in four Gerrit projects

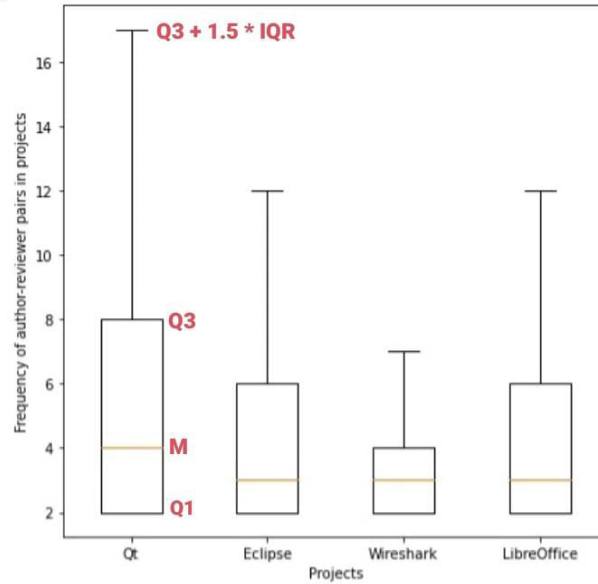


Figure 4.2 exhibits box plots of pair frequencies in four projects without outliers. The threshold to detect review buddies smell is $Q3 + 1.5 * IQR$. This is a dynamic threshold that differ in chosen projects.

Table 4.9 shows the number of pairs, frequency of review buddies, and smell

Table 4.9: Total number of author-reviewer pairs and review buddies smell in Gerrit projects.

Projects	# of author-reviewer pairs	Review buddies frequency	Smell perc. %
Qt	7,311	350	4.7
Eclipse	2,859	156	5.45
Wireshark	1,183	51	4.31
LibreOffice	1,406	43	3.05

Table 4.10: Average of LOCs reviewed by reviewers of high frequent pairs (review buddies) and low frequent pairs (non-review buddies) in four Gerrit projects.

Projects	Average of LOCs by reviewers of high frequent pairs	Average of LOCs by reviewers of low frequent pairs
Qt	16.6k	2.1k
Eclipse	18.1k	2.6k
Wireshark	3.6k	1.5k
LibreOffice	5.7k	0.7k

ratio in four OSS projects. Figure 4.3 displays the difference between the workload of reviewers considered review buddies and standard reviewers. As illustrated in Figure 4.3, review buddies' workload has a broader range, higher median value, and higher maximum value than the workload of standard reviewers.

In addition to box plots and median values, Table 4.10 shows the average of LOCs reviewed by a reviewer of high frequent pairs (review buddies) vs. low frequent pairs (non-review buddies). As shown in Table 4.10, the average of LOCs for high frequent pairs is more than the average of LOCs for low frequent pairs in all projects. Figure 4.4 shows four scattered plots (one for each project) as the distribution of high and low frequent pairs and the CBCS score of reviewer of a pair in four projects. In all of these plots in Figure 4.4, outliers were discarded. Outliers are the pairs that LOCs committed by the author and reviewed by the reviewer of the pair is unfeasibly high with significant difference compared to other pairs. In these plots in Figure 4.4, the X-axis represents the occurrence frequency of pairs in the code review data of the project, and the Y-axis shows

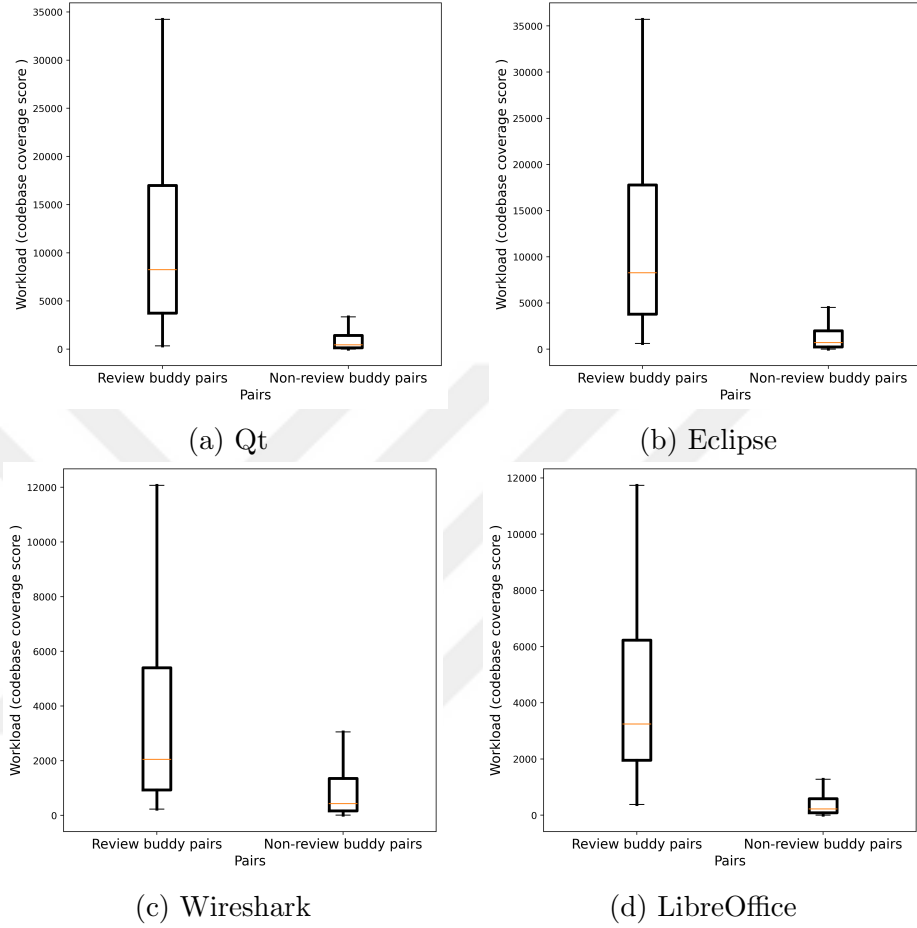


Figure 4.3: Distribution of review workload (CBCS score, reviewed LOCs by reviewers of pairs) in two different groups, review buddy pairs, and non-review buddy pairs

the sum of LOCs committed by the author and reviewed by the reviewer of a particular pair (CBCS score). Orange dots show low frequent pairs (non-review buddies), and blue dots show high frequent pairs (review buddies). Two green horizontal lines in each plot show the average LOCs reviewed by reviewers in each category. As illustrated, the green line related to high frequent pairs has a higher value than the low frequent pairs in all four projects.

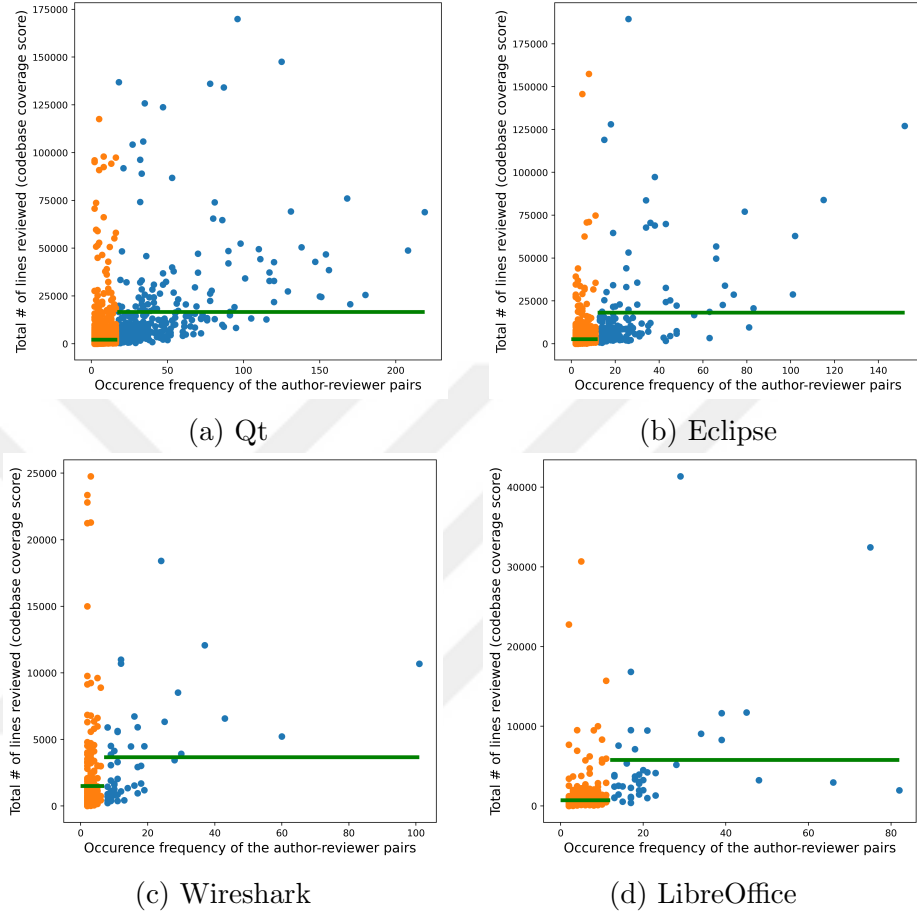


Figure 4.4: Distribution of frequency and workload of high-frequency pairs (blue dots as review buddies) and low-frequency pairs (orange dots as non-review buddies)

4.4.2 Reopened issues impact

In an ideal code review process, each PR should be linked to a related issue if the code review tool is integrated with an issue tracking tool. Among our chosen projects, the code review system (Gerrit) is integrated with Jira in the Qt project and Bugzilla in Eclipse, LibreOffice, and Wireshark projects. While the other systems are still accessible, the issue tracking and code review systems of the Wireshark project were migrated to GitLab in August 2020. Their Bugzilla instance is available, but their Gerrit instance is not accessible anymore. We previously had retrieved the code review instances of the Wireshark project before the project was migrated to GitLab, but these instances are no longer available.

Table 4.11: Distribution of the smelly PRs linked to reopened issues for bypassed negative reviewer smell, review buddies smell and lack of code review smell in four Gerrit projects

Projects	Merged PRs linked to reopened issues	Bypassed negative reviewer PRs linked to reopened issues	Review buddies PRs linked to reopened issues	LOCR PRs linked to reopened issues
Qt	418	2 (0.47%)	190 (45.23%)	3 (0.71%)
Eclipse	3276	5 (0.15%)	47 (1.32%)	1,512 (46.15%)
Wireshark	165	0 (0%)	16 (9.30%)	76 (46.06%)
LibreOffice	719	1 (0.13%)	36 (5.00%)	325 (45.20%)

Table 4.12: Distribution of the smelly PRs linked to reopened issues for ping pong smell, sleeping review smell and large changeset smell in four Gerrit projects

Projects	Total PRs linked to reopened issues	Ping pong PRs linked to reopened issues	Sleeping PRs linked to reopened issues	PRs with LCHS linked to reopened issues
Qt	420	199 (47.38%)	253 (60.23%)	21 (5.0%)
Eclipse	3,541	156 (4.40%)	1120 (31.62%)	691 (19.51%)
Wireshark	172	11 (6.39%)	28 (16.27%)	11 (6.39%)
LibreOffice	719	38 (5.28%)	198 (27.53%)	51 (7.09%)

To obtain reopen ratio for the bypassed negative reviewer, review buddies, and LOCR smells, the number of smelly PRs linked to reopened issues is divided by the number of merged PRs linked to reopened issues since these smells occur in only merged PRs based on their definition. On the other hand, to obtain reopen ratio for ping pong, sleeping and LCHS smells, the number of smelly PRs linked to reopened issues is divided by the total number of PRs linked to reopened issues consisting of abandoned and merged PRs. Tables 4.11 and 4.12 illustrate the ratio of PRs linked to reopened issues in six code review smells. For instance, 46.15% of the lack of code review PRs in the Eclipse project are linked to the issues in Bugzilla that went through reopen status once in their status change history. This ratio is 46.06% for the Wireshark project and 45.20% for the LibreOffice project. We detected only 3 PRs that suffer from LOCR smell and are linked to reopened issues in the whole Qt project because the Qt project warns its developers clearly

to prevent self-reviewed and un-reviewed PRs in Qt contribution guideline [24]. In contrast, the emphasis on the lack of code review is not explicit in the other three projects [25] [26] [27]. We quantitatively analyzed reopen ratio in the existence of all mentioned smells. Among all mentioned smells, in all projects except Qt, lack of code review PRs have the highest ratio to cause the linked issues (linked to the same PR) to reopen and surface again in the future. Since the PRs with lack of code review smell are not reviewed or are self-reviewed, the potential problems may not be solved before merging, leading to the related issue's reopening later. On the other hand, since in the Qt project, 44.40% of merged or abandoned PRs experience sleeping review smell, it is expected that sleeping PRs have the highest ratio of issue reopening (60.23%). In Eclipse, Wireshark, and LibreOffice projects, sleeping smell has the second-highest ratio to have reopened issues impact. Also, ping pong and large changeset smells have the third and the fourth-highest ratio of reopened impact, respectively. Review buddies smell may also cause reopened issues in the projects. The behavior of review buddies may lead to inattentive code reviews, and potential problems may not be solved before merging. The ratio of PRs with review buddies smell and linked to reopened issues is the highest in the Qt project. The main reason for this high ratio is that the Qt project is the only project whose contribution guideline warns developers about review buddies smell [24] and the occurrence of this smell has high bug reopen ratio as a consequence of this smell. Another smell that can lead to an increment in reopen ratio is bypassed negative reviewer smell. In this smell, the negative reviewer may have reviewed the PR properly and rejected it for proper reasons, but positive reviewers may have reviewed the commit superficially, leading to the PR merging and linked issues' reopening.

4.4.3 Time to complete review (TTCR) impact

This impact shows the time to complete review (TTCR) in smelly and normal (non-smelly) reviews. Smells such as (1) sleeping reviews, (2) bypassed negative reviewer, (3) reviewer-author ping pong, (4) large changesets (LCHS), and (5)

Table 4.13: Average TTCR of smelly and non smelly PRs (in days)

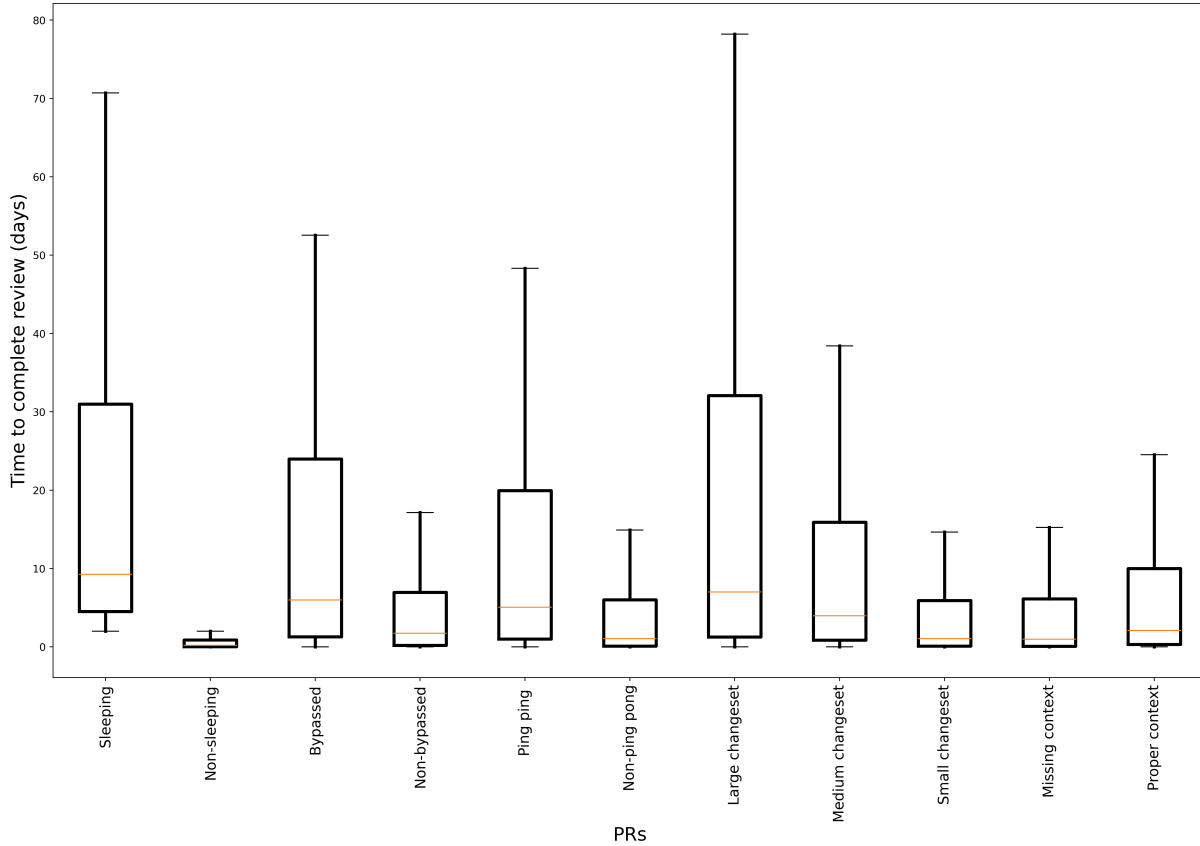
Projects	Sleeping	Non-sleeping	Bypassed	Non-bypassed	Ping pong	Non-ping pong
Qt	57.39	0.50	58.80	18.35	38.33	24.04
Eclipse	86.38	0.33	33.23	11.74	54.24	24.97
Wireshark	54.57	0.26	10.88	4.76	19.23	6.78
LibreOffice	21.77	0.41	10.60	4.38	13.76	5.13

Table 4.14: Average TTCR of smelly and non smelly PRs (in days)

Projects	Large CHS	Medium CHS	Small CHS	Missing context	Proper context
Qt	56.97	42.22	19.56	21.57	30.29
Eclipse	42.33	33.59	18.76	16.28	31.97
Wireshark	16.67	11.19	5.60	7.47	8.20
LibreOffice	8.51	7.02	4.39	3.53	6.60

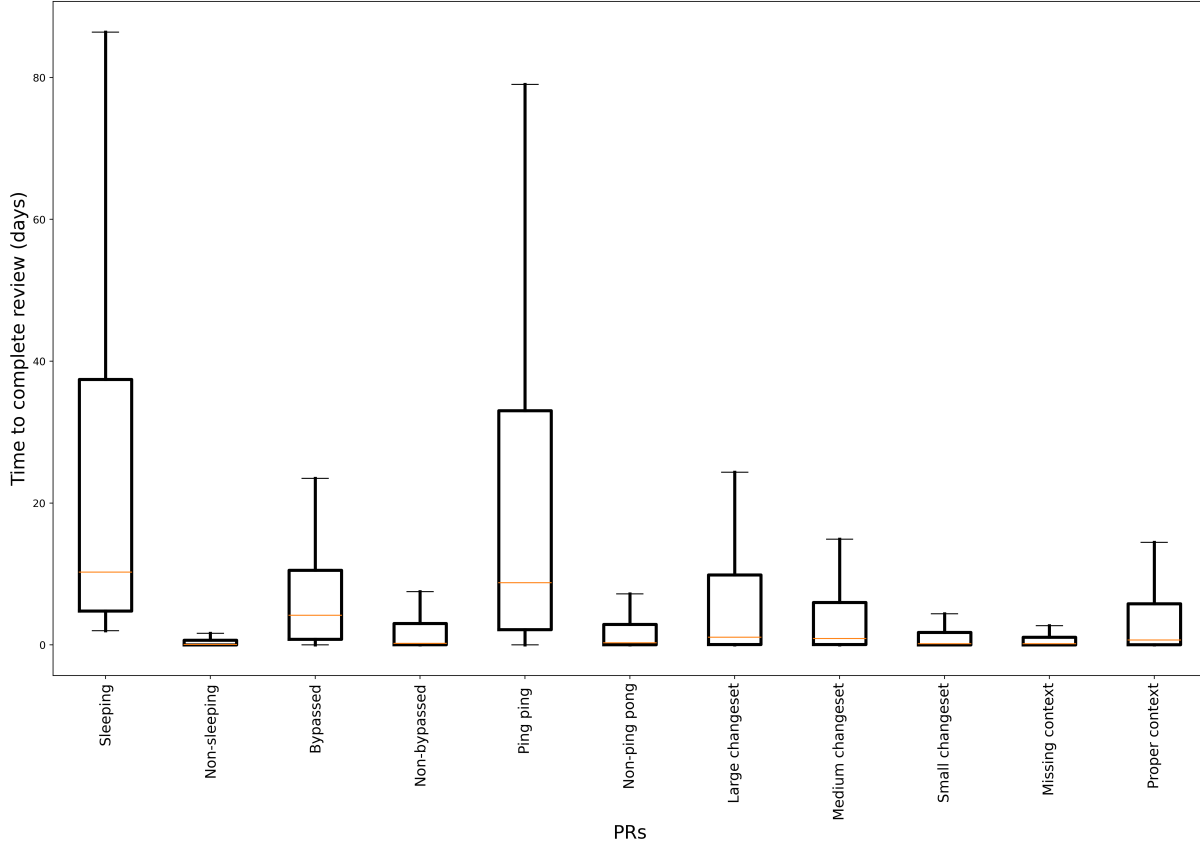
missing context can affect the time to complete review. Figures 4.5, 4.6, 4.7 and 4.8 illustrate the box plots related to the distribution of time to complete review of smelly and non-smelly PRs for five types of smells (sleeping review, bypassed negative reviewer, ping pong review, large changeset, and missing context smells) in our chosen projects, respectively. These figures illustrate that TTCR of smelly PRs has a broader range and higher median value in sleeping review, bypassed negative reviewer, and ping pong review smells in all four projects. For large changeset smell, distribution of TTCR of smelly PRs has a broader range and higher median value. However, the median values in large, medium, and small changesets differ slightly. In contrast to other smell types, for missing context smell, the distribution of TTCR of smelly PRs is more centered and has a tighter range, lower median value, and even lower maximum value. Also, a positive skewness exists in the distribution of TTCR of smelly PRs and non-smelly PRs downward, toward zero in all smell types. Note that explained box plots are visualized without considering outliers. Tables 4.13 and 4.14 show the average TTCR value (mean value) in smelly and non-smelly PRs for five smell categories. In all cases except missing context smell, smelly PRs have a higher average TTCR than non-smelly (proper) PRs.

Figure 4.5: Distribution of time to complete review of smelly and non-smelly PRs for five types of smells in Qt project



Sleeping reviews have a higher average TTCR than non-sleeping reviews, with a significant difference between average TTCR values. We realized that bypassed reviews might take more time to be completed because of conflicts during the review process (negative reviewer, re-assigning new reviewer as positive reviewer). Furthermore, ping pong smell can lead to higher average TTCR due to excessive iterations in the ping pong smell. The average TTCR for PRs with large changesets is higher than for those with medium and small changesets, and the average TTCR of PRs increases with the increment of the size of a changeset. Furthermore, we expected that due to missing linkage and lack of proper description in missing context smell, missing context PRs may require higher TTCR in Gerrit

Figure 4.6: Distribution of time to complete review of smelly and non-smelly PRs for five types of smells in Eclipse project



projects, but we observed that the TTCR of missing context PRs is less than the TTCR of normal, proper PRs (non-missing context PRs). We justify this situation with the average of the number of inline and summary comments (sum of both inline and summary comments provided by reviewers) in missing context and proper context PRs. We observed that in all four projects, the average and median values of the number of inline and summary comments in missing context PRs are less than proper context PRs, and as a consequence, the time to complete review of missing context PRs is less than the TTCR of proper context PRs.

Figure 4.7: Distribution of time to complete review of smelly and non-smelly PRs for five types of smells in Wireshark project

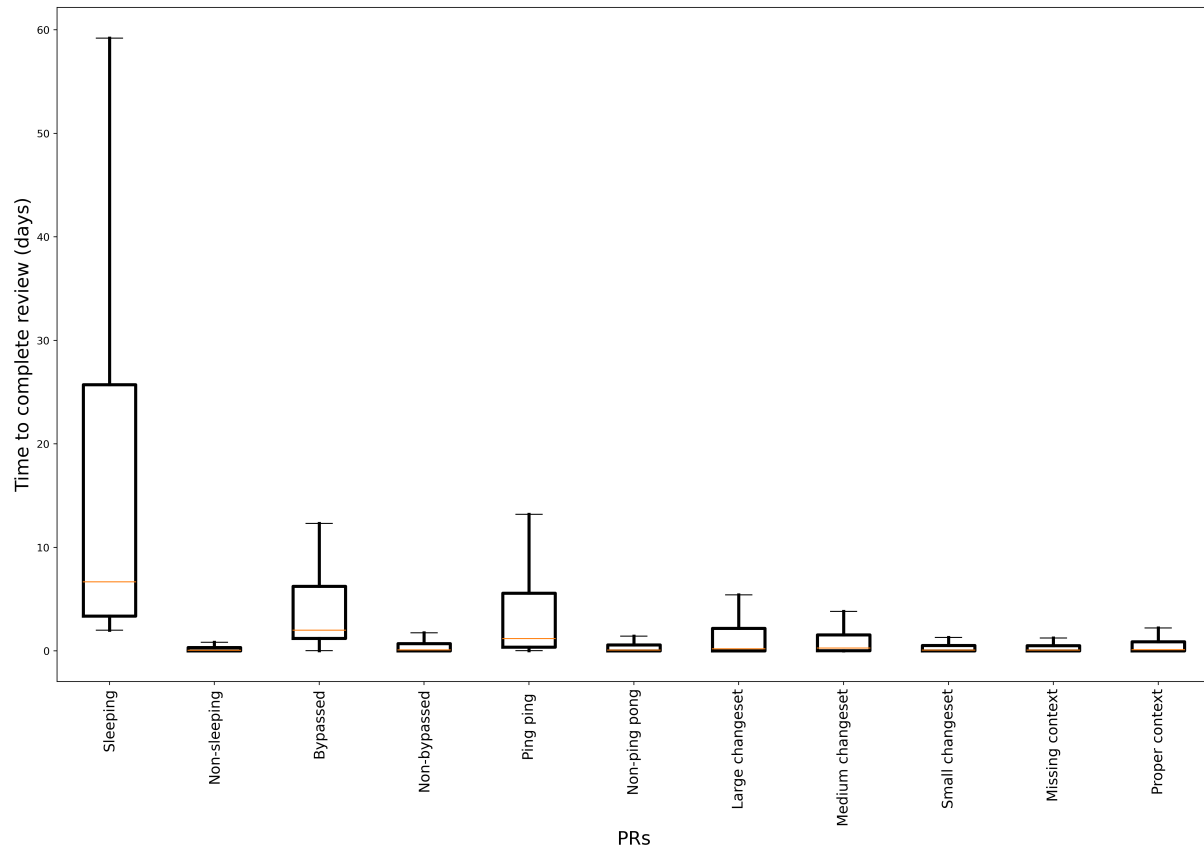
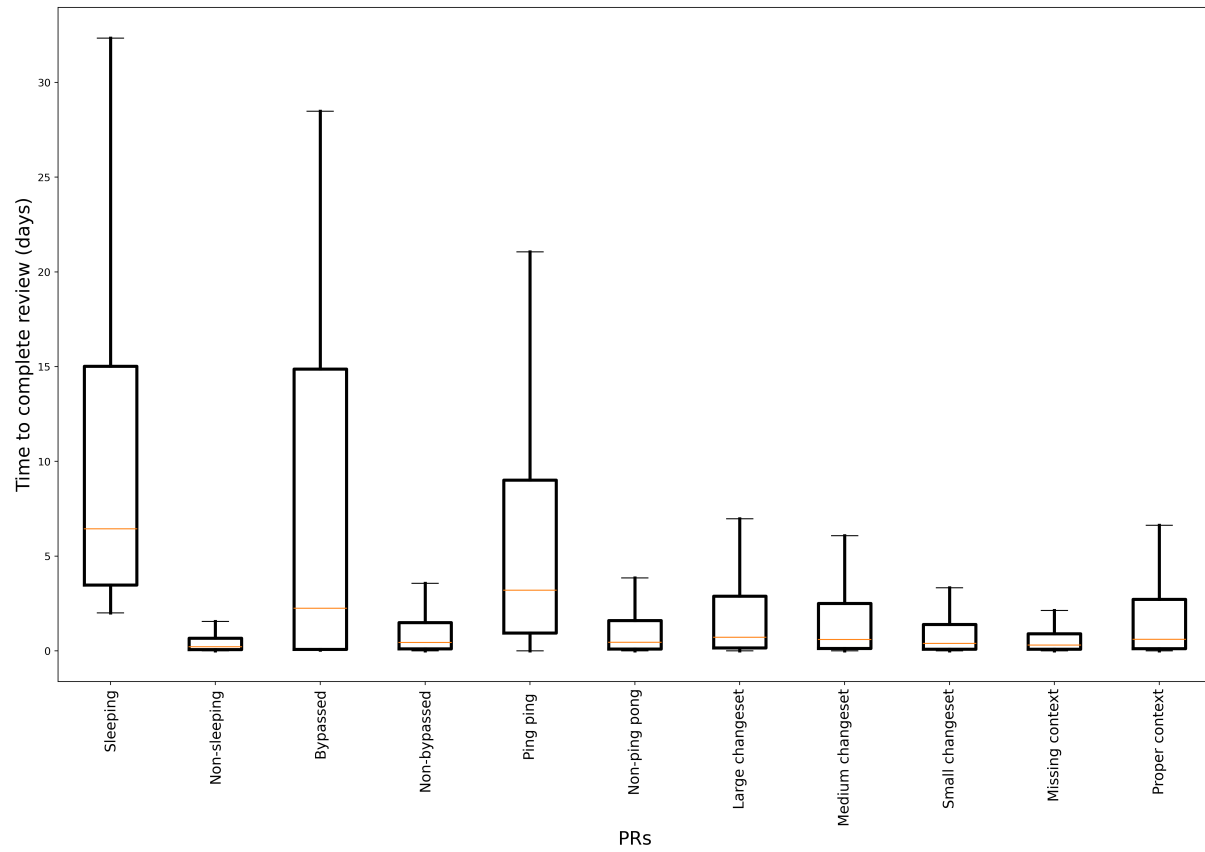


Figure 4.8: Distribution of time to complete review of smelly and non-smelly PRs for five types of smells in LibreOffice project



4.4.4 Abandoned pull requests impact

As we discussed, challenges and difficulties in the code review process may lead to abandoned PRs. Smells such as (1) ping pong smell, (2) sleeping review smell, (3) large changesets smell, and (4) missing context smell may lead to abandoned PRs. PRs with the ping pong smell may get abandoned as the author gets frustrated with the constant feedback and modification iterations. Furthermore, long elapsed time in sleeping reviews may disturb or even block the development process and make the author abandon the PR. Also, the reason behind the abandoned pull requests impact in large changesets smell may be due to a proper and attentive code review which requires a long review completion time and more iterations in the review cycle for PRs, which may make the review process frustrating for the author. The last smell that may lead to abandoned PRs is the missing context smell. Due to the missing information that these smelly PRs experience, reviewers of these PRs may request needed information during several iterations, and reviewing time and number of iterations may increase. These challenges may cause the author to abandon the PR, and abandoned pull requests impact may occur. Note that the only reason behind abandoning a PR is not necessarily the occurrence of a smell. There is a possibility that the reason for abandoning a PR is not because of one of the code review process smells but because the PR content (e.g., source code) is problematic. To confirm the relationship between a PR being smelly and PR getting abandoned, we also checked other cases. We checked abandoned PRs without a particular smell (single-negative), PRs which are not abandoned with a particular smell (single-negative), and without a particular smell (double-negative). Tables 4.15 and 4.16 show the frequency of four smell categories and abandonment impact in these smelly reviews. For instance, in the Qt project, 3.82% of the ping pong smells get abandoned even after three or more iterations. Furthermore, 13-19% of the sleeping reviews get abandoned in all four projects. Also, the abandonment ratio among missing context PRs is between 6-24% in all four projects. By comparing abandoned pull requests impact ratio from Tables 4.15 and 4.16, we observe that in all four projects, ping pong reviews are less prone to get abandoned than other smelly PRs.

Table 4.15: Abandoned pull requests impact of two smell categories: ping pong and sleeping review smells in four Gerrit projects

Projects	Ping pong Smell Freq.	Impact Freq.	Impact Perc. %	Sleeping re-view Smell Freq.	Impact Freq.	Impact Perc. %
Qt	23,479	897	3.82	40,958	7,113	17.36
Eclipse	3,572	120	3.35	21,166	4,142	19.56
Wireshark	1,672	30	1.79	2,459	343	13.94
LibreOffice	1,372	18	1.31	13,437	1,855	13.80

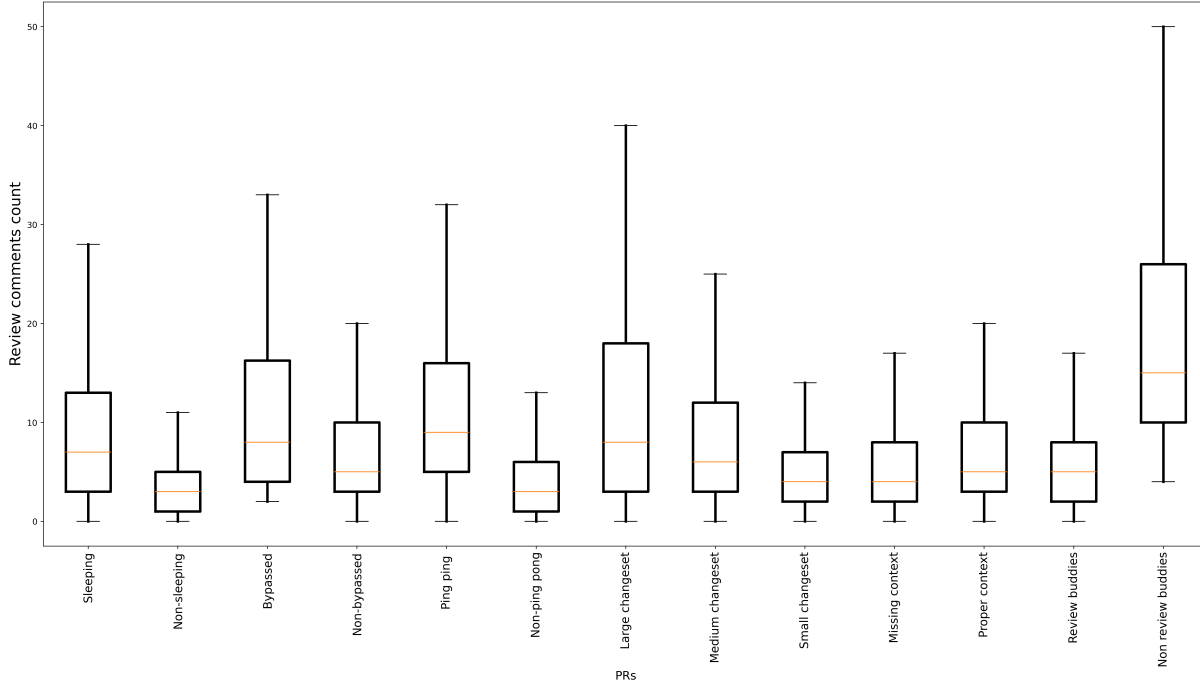
Table 4.16: Abandoned pull requests impact of two smell categories: LCHS and missing context smells in four Gerrit projects

Projects	LCHS Smell Freq.	Impact Freq.	Impact Perc. %	Missing con-text Smell Freq.	Impact Freq.	Impact Perc. %
Qt	9,893	1,996	20.17	22,260	3,868	17.37
Eclipse	13,845	2,384	17.21	24,439	5,979	24.46
Wireshark	2,285	170	7.43	5,170	357	6.90
LibreOffice	6,345	637	10.03	24,088	1,835	7.61

4.4.5 Number of review comments impact

One of the critical features of code review is the review comments that are given as feedback by reviewers. Reviewers can comment on the entire pull request, a particular file (file-level comments), or specific lines of code in a file (inline comments) to generate discussion relevant to code review. The existence of code review smells in a pull request can affect the number of review comments reviewers give. The number of review comments in a PR review that takes a long time to be completed (sleeping review smell) might be more than PRs without sleeping review smell. Moreover, bypassed negative reviewer smell may affect the number of review comments. Furthermore, based on the definition of ping pong review smell (more than three feedback and modification iterations), we expected the number of review comments in ping pong reviews is higher than the number of review comments in PRs without ping pong review smell. Also, reviewing, approving, and merging a large changeset is a difficult task, and proper review of

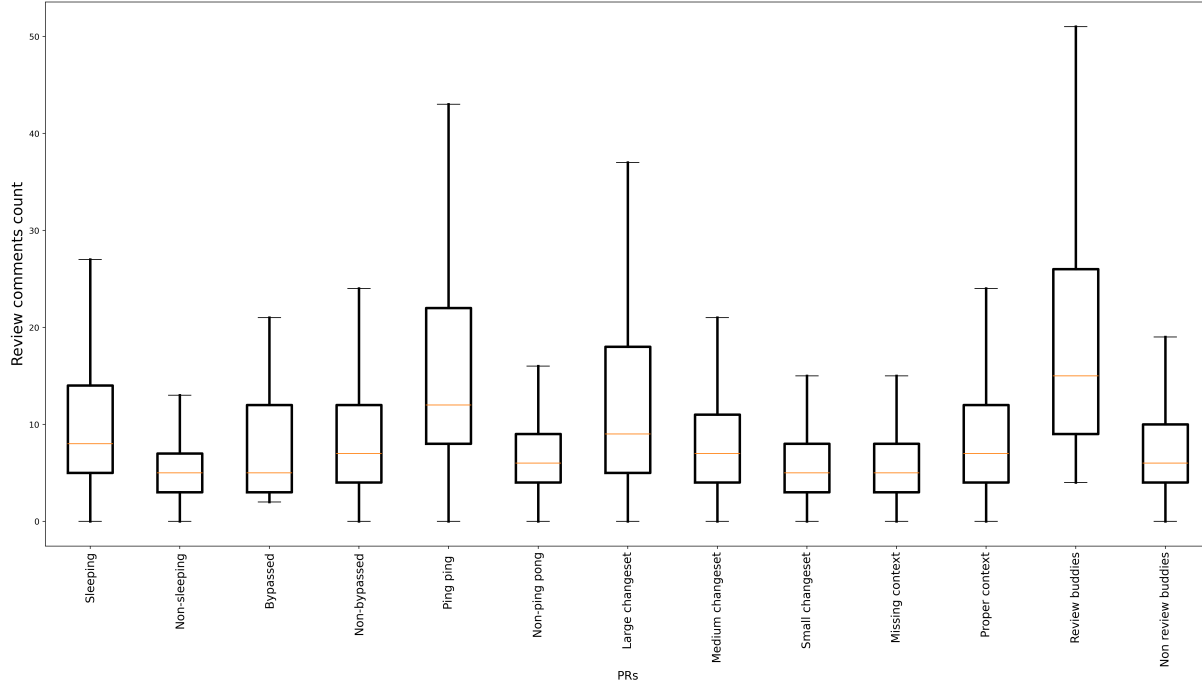
Figure 4.9: Distribution of number of review comments of smelly and non-smelly PRs for six types of smells in Qt project



large changesets may change the number of review comments. Furthermore, based on the definition of missing context smell, reviewing such PR may require more review comments for requesting more information about the content of the PR. Based on definition of review buddies smell, existence of this smell may change the number of review comments since the reviewer is assigned consistently to review the same author's PRs, and the reviewer has a good acquaintanceship with the author. Figures 4.9, 4.10, 4.11, and 4.12 display the distribution of number of review comments in six type of smells.

The distribution of the number of review comments in sleeping reviews has a wider range and higher median value, and a similar skewness to the distribution of the number of review comments in non-sleeping reviews. This skewness is slight and similar in all projects except, in the Qt project, the skewness of distribution in non-sleeping reviews is more positive (downward skewness, toward zero).

Figure 4.10: Distribution of number of review comments of smelly and non-smelly PRs for six types of smells in Eclipse project

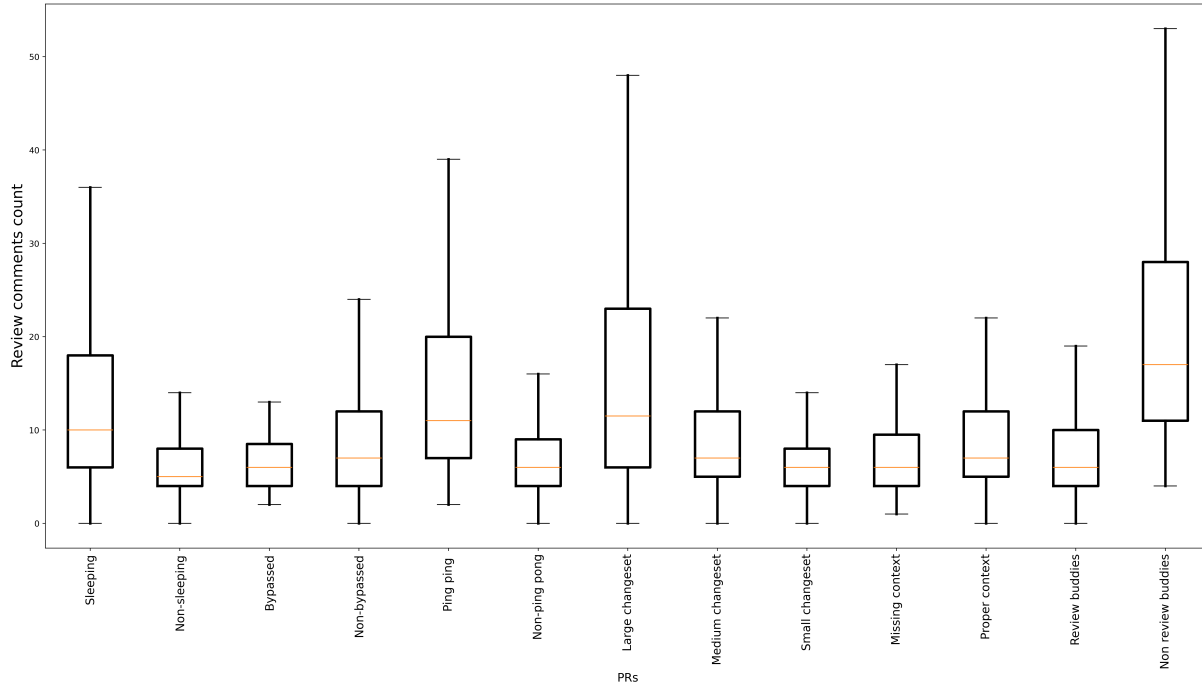


The distribution of the number of review comments in bypassed negative reviewer smell has a higher median value in Qt project and lower median value in Eclipse and Wireshark projects. In LibreOffice, same median value in smelly and non-smelly PRs is observed.

The distribution of the number of review comments in ping pong reviews has a broader range, but the skewness of the distribution is very similar to the distribution of the number of review comments in PRs without ping pong review smell. Moreover, the median value in smelly PRs is higher than non-smelly PRs in ping pong smell.

Among PRs with different changeset size categories, median value of distribution of review comments decreases with size, and also the skewness of these distributions is similar, with positive skewness in all three categories.

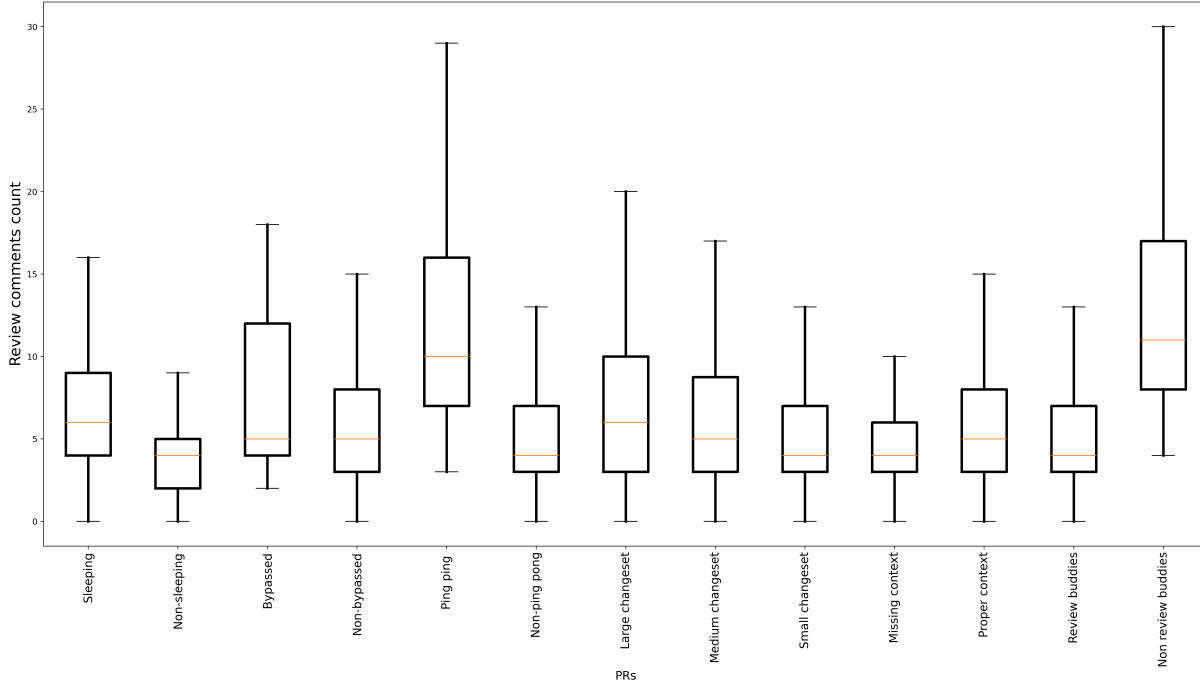
Figure 4.11: Distribution of number of review comments of smelly and non-smelly PRs for six types of smells in Wireshark project



Moreover, related to the number of review comments in PRs with missing context smell, the median value of smelly PRs is less than the median value of non-smelly PRs in all four projects. Distribution of the number of comments has a tighter range in PRs with missing context.

In addition, in Qt, Wireshark, and LibreOffice projects, distribution of number of review comments in PRs with review buddies smell has a tighter range, and the median value related to smelly PRs is lower than the median value of non-smelly PRs. In the Eclipse project, distribution of the number of review comments in PRs with review buddies smell is more expansive, and the median value is higher compared to non-smelly instances.

Figure 4.12: Distribution of number of review comments of smelly and non-smelly PRs for six types of smells in LibreOffice project



4.4.6 Contribution of large changeset reviewers impact

As we mentioned in Section 3, a reviewer's reviewed lines of code and received knowledge increase dramatically by reviewing a large changeset. Our goal is to analyze large changeset reviews' contribution in the future review tasks of PRs with large changesets. By gaining this knowledge and code/file ownership because of reviewing large changesets, we expect reviewers to contribute more in upcoming PR reviews. However, we cannot see a pattern that justifies the increased contribution of large changeset reviewers. We divided the code review history (sorted based on the creation time of PRs) into ten subsets and calculated the ratio of reviewed large PRs of each reviewer in each subset. Figure 4.13 shows the pattern of the top 3 reviewers' contribution (most active reviewers) in reviewing large PRs in ten subsets. In Figure 4.13, each plot shows the result of a

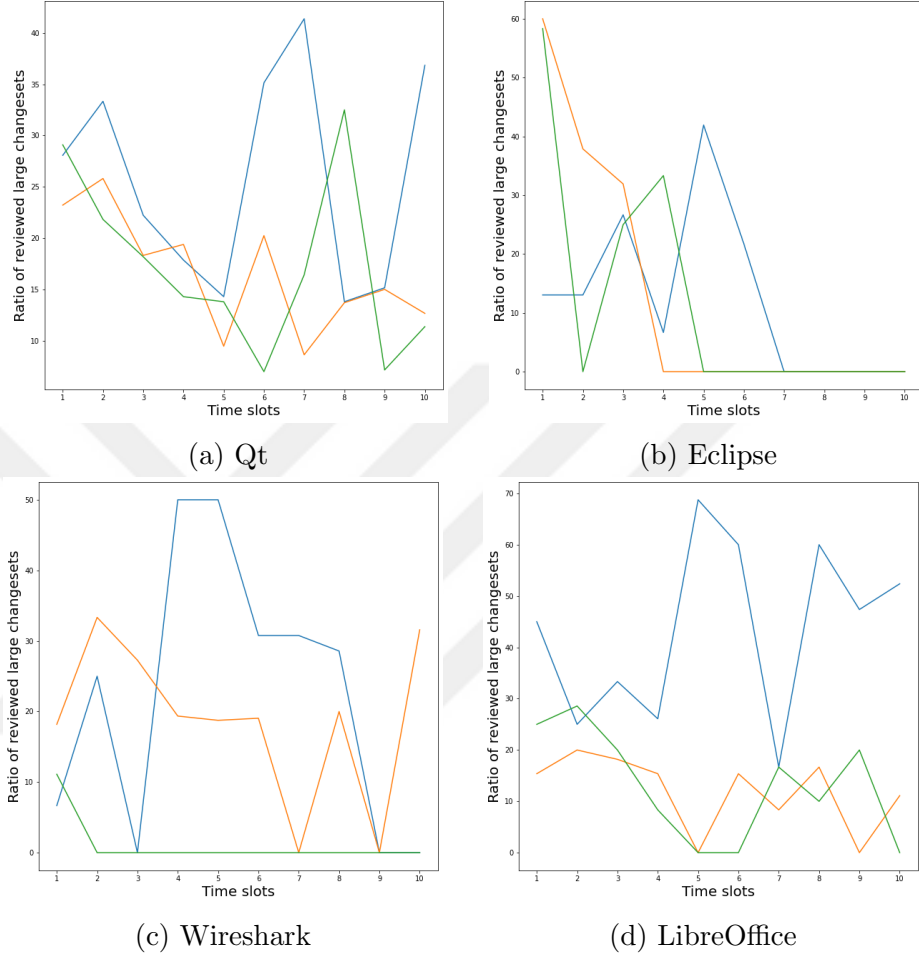


Figure 4.13: Pattern of large changeset reviewers' contribution (ratio of reviewed PRs with large changesets) over time in four Gerrit projects

project. In these plots, each colored line shows the ratio pattern of one of the top 3 reviewers in reviewing large PRs over time in code review history. With these results, we are still unable to detect the impact of large changeset reviews on future contributions of reviewers. This might be attributable to various reasons, such as the decisions we make in our research setup, and more research is needed to confirm or refute such a finding. For instance, Figure 4.13, part b, shows the ratio of reviewed large PRs of top 3 reviewers (most active reviewers) in Eclipse project. all three reviewers' contribution ratio is zero after 7th subset, but they do not follow a repetitive pattern, and we cannot see a consistent increase in their contribution. This statement is true in other Gerrit projects used in our analysis.

Chapter 5

Hypothesis Testing

To analyze the significance of our results, we perform different statistical tests based on their usage. To statistically analyze the co-occurrence of a committers' side smell and a reviewers' side smell at the same PR and discover the correlation between smells, we perform *Fisher's exact* test, which produces exact and precise p-values. *Fisher's exact* test provides an alternative to *Pearson's r* test when using nominal data type instead of numeric variables. A nominal data type is a type of data used to label variables without providing any quantitative value (e.g., for each PR in a project, having a specific smell type is True or False). We perform *Fisher's exact* test to verify whether the proportion of a reviewers' side smell is larger in a PRs with committers' side smell than in other PRs, and we use the odds ratio to measure this proportion. We use the *Fisher's exact* test to inquire about the significance of the results of correlations between smells reported in Section 4.3. We also used *Fisher's exact* to analyze the significance of results related to reopened issues impact reported in Section 4.4.2 and abandoned pull requests impact reported in Section 4.4.4. Next, we perform *Mann-Whitney U* test on the results of (1) time to complete review (TTCR) impact, (2) shared knowledge impact, and (3) number of review comments impact. *Mann-Whitney U* test is used to compare whether there is a difference in the dependent variable for two independent groups. Finally, we applied *chi-square* test to compare more than two independent groups. *Chi-square* test inquiries the relationship between a

PR being smelly and the likelihood of reopened issues or abandoned pull requests impacts. In this test, the type of smell in a PR is not essential, and the test is applied to different smell types to analyze whether a PR being smelly is likely to create reopened issues or abandoned pull requests impacts.

5.1 Significance of Results in Correlation Between Smells, Reopened Issues Impact, and Abandoned Pull Requests Impact

To statistically analyze (1) the co-occurrence of a committers' side smell and a reviewers' side smell at the same PR and discover the correlation between smells, (2) correlation between smelly PRs and reopened issues, and (3) correlation between smelly PRs and abandoned pull requests impact, we apply Fisher's exact test to obtained results. Fisher's exact test is performed to see a significant relationship between two categorical variables (e.g., PRs with bypassed negative reviewer smell and PRs with sleeping review smell). We applied Fisher's exact test to confirm that the correlations between discussed smells are proved to be a significant relationship and not only co-occurrence of two different smells in a PR. Table 5.1 displays the frequency of PRs with or without bypassed negative reviewer smell and sleeping smell in the Eclipse project. For instance, in the Eclipse project, we observed 66 PRs with both mentioned smells (in the same PR). Also, the total value of each row and column of Table 5.1 and the grand total are displayed in this table's margins. Fisher's exact test aims to investigate if there are smell-related differences between these four categories illustrated in Table 5.1. With this test, we can confirm that these observations were not solely due to chance.

Null Hypothesis 1 *Occurrences of bypassed negative reviewer smell and sleeping review smell in the same PR are statistically independent.*

To calculate the p-value of Fisher's exact test, first, we determine all possible

tables with fixed marginal values and their respective probabilities given our fixed margins. For each possible table, the table probability is calculated as:

$$p(a) = \frac{\binom{a+c}{a} \binom{b+d}{b}}{\binom{a+b+c+d}{a+b}} \quad (5.1)$$

Finally, the p-value is calculated as the sum of the probabilities for all possible tables having a probability equal to or smaller than the observed values (Values in Table 5.1). If the achieved p-value is less than 0.05, which is our chosen significance level, the null hypothesis would be rejected and there would be a statistically significant association and correlation between smells. Otherwise, the two smells are not correlated and only co-occurred in the same PR. Furthermore, Fisher's exact test can calculate the odds ratio in each case. The odds ratio reveals the likelihood of positive cases occurring more than negative cases. For instance, in Table 5.1 Positive cases are PRs with both bypassed negative reviewer smell and sleeping review smell (a), and also PRs without any of these smells (d). On the other hand, negative cases are PRs with only one of these smells (b and c). For instance, the odd ratio equal to 10, shows odds of observing positive cases are 10 times more than observing negative cases in the Eclipse project.

Table 5.2 shows the p-values of Fisher's exact test for all the correlations mentioned in Section 4.3 in four projects. In each case, if the calculated P-value is less than 0.05 (significant level), the null hypothesis is rejected, and associated smells are correlated in the associated project. The coloring scheme is used to show the expected and unexpected statistical results. The red color illustrates unexpected results where the correlation is not significant and associated smells just co-occurred in the particular project. In other words, occurrences of two associated smells are statistically independent. Furthermore, Table 5.3 shows the odds ratio of each case in four projects. Note that the odds ratio can also be less than 1, which shows negative cases occur more than positive cases.

Main deduction: Bypassed negative reviewer smell is correlated with sleeping review smell and ping pong review smell in all projects.

Large changeset smell is correlated with sleeping review smell in all projects.

Large changeset smell is correlated with ping pong smell in all projects except LibreOffice project. We could not observe a strong correlation between LCHS and ping pong review smells in LibreOffice project.

Missing context smell is correlated with sleeping review and ping pong smells but odds ratio of both correlations is less than 1 in all projects thus negative cases are more likely than positive cases.

Tables 5.4 and 5.5 illustrate the p-value and odds ratio of the results related to reopened issues impact. The red color in p-values illustrates unexpected results where a particular smell type is not prone to get the linked issue reopened. The red color in odds ratios displays values less than 1 where negative cases occur more than positive cases. For instance, ping pong and large changeset smells have significant relationships with reopened issues impact in all four projects.

Main deduction: Ping pong smell and large changest smell are significantly likely to get linked issues reopened.

Results of reopened issues impact are significant for ping pong review smell and large changeset smell in all four projects.

Results of reopened issues impact are significant for Lack of code reviewer smell and sleeping review smell in Qt and LibreOffice projects.

Results of reopened issues impact are significant for review buddies smell in only Qt project.

Tables 5.6 and 5.7 illustrate the p-values and odd ratios related to the results of abandoned pull requests impact. None of the p-values are greater than 0.05 (significance level), which confirms that ping pong review smell, sleeping review smell, large changeset smell, and missing context smell are likely to lead to abandoned pull requests. Moreover, except for the Qt project for missing context smell and the other three projects for ping pong review smell, the odds ratio is more than 1 in all cases, which shows positive cases occur more than negative

cases in our chosen projects .

Main deduction: Sleeping review, large changesets and missing context smells are likely to cause the PR under review to get abandoned. The relationship between a PR being smelly and get abandoned is significant with odds ratio more than 1 in sleeping review, large changesets and missing context smells in all four projects.

Ping pong smell and abandoned PRs also have a significant relationship. However odds ratio of this relationship is less than 1 in all four projects.

Table 5.1: Frequency of PRs with or without bypassed negative reviewer smell and sleeping review smell in Eclipse project

Smells	PRs with Sleeping review smell	PRs without sleeping review smell	Total
PRs with bypassed negative reviewer smell	a = 66 (positive case)	b = 48 (negative case)	a + b = 114
PRs without bypassed negative reviewer smell	c = 16,958 (negative case)	d = 41,576 (positive case)	c + d = 58,534
Total	a + c = 17,024	b + d = 41,624	a + b + d + c = 58,648 (Grand total)

Table 5.2: P-values of Fisher's exact test for correlation between different smells in four Gerrit projects. Expected Results (Blue Color), Unexpected Results (Red Color)

Projects	Bypassed & sleeping	Bypassed & ping pong	LCBS & sleeping	LCBS & ping pong	Missing & sleeping	Missing & ping pong
Qt	1.94e-07	1.10e-39	0.0	0.0	1.45e-143	1.05e-314
Eclipse	1.45e-10	7.58e-07	6.32e-291	1.53e-123	0.0	6.73e-119
Wireshark	2.30e-09	4.16e-30	1.51e-54	5.51e-11	6.85e-11	9.29e-05
LibreOffice	8.34e-08	1.30e-05	2.28e-35	0.053	4.19e-312	2.27e-92

Table 5.3: Odds ratio of Fisher's exact test for correlation between different smells in four Gerrit projects. Expected Results (Blue Color), Unexpected Results (Red Color)

Projects	Bypassed & sleeping	Bypassed & ping pong	LCBS & sleeping	LCBS & ping pong	Missing & sleeping	Missing & ping pong
Qt	2.47	12.15	2.84	2.37	0.66	0.49
Eclipse	3.37	3.84	2.06	2.44	0.44	0.38
Wireshark	6.51	38.75	2.41	1.57	0.72	0.79
LibreOffice	4.2	7.21	1.45	0.83	0.45	0.26

Table 5.4: P-values of Fisher’s exact test for reopen impact in four Gerrit projects. Expected Results (Blue Color), Unexpected Results (Red Color)

Projects	Bypassed & Reopen	Review buddies & Reopen	LOCR & Reopen	Ping pong & Reopen	Sleeping & Reopen	LCHS & Reopen
Qt	0.19	1.77e-06	0.00022	1.09e-17	1.13e-06	3.97e-06
Eclipse	0.83	0.81	0.10	0.04	0.09	0.60
Wireshark	1.0	0.70	0.87	1.13e-36	0.44	0.006
LibreOffice	0.53	0.86	5.97e-12	4.94e-06	0.0049	0.0007

Table 5.5: Odds ratio of Fisher’s exact test for reopen impact in four Gerrit projects. Expected Results (Blue Color), Unexpected Results (Red Color)

Projects	Bypassed & Reopen	Review buddies & Reopen	LOCR & Reopen	Ping pong & Reopen	Sleeping & Reopen	LCHS & Reopen
Qt	2.47	1.61	0.18	2.36	1.62	0.39
Eclipse	0.77	1.02	0.94	0.84	1.06	0.97
Wireshark	0	0.86	0.96	0.63	1.17	0.44
LibreOffice	1.30	0.95	0.59	2.35	1.27	0.62

Table 5.6: P-values of Fisher’s exact test for abandoned PRs impact in four Gerrit projects. Expected Results (Blue Color), Unexpected Results (Red Color)

Projects	Ping pong & abandoned	Sleeping & abandoned	LCHS & abandoned	Missing & abandoned
Qt	0.0	1.31e-140	1.87e-66	4.27e-53
Eclipse	5.55e-133	2.14e-69	6.28e-07	0.0
Wireshark	3.87e-14	4.23e-73	2.23e-06	1.28e-09
LibreOffice	5.13e-24	1.09e-228	1.95e-20	2.33e-05

Table 5.7: Odds ratio of Fisher’s exact test for abandoned PRs impact in four Gerrit projects. Expected Results (Blue Color), Unexpected Results (Red Color)

Projects	Ping pong & abandoned	Sleeping & abandoned	LCHS & abandoned	Missing & abandoned
Qt	0.17	1.64	1.62	1.39
Eclipse	0.17	1.47	1.13	2.57
Wireshark	0.30	4.04	1.54	1.53
LibreOffice	0.17	3.00	1.54	1.14

5.2 TTCR, Review Workload, and Number of Review Comments VS. Process Smells

To analyze the significance of results of (1) time to complete review (TTCR) impact, (2) shared knowledge impact, and (3) number of review comments impact, we apply Mann-Whitney U test to achieved results. Mann-Whitney U test is a non-parametric test for independent samples in which data does not have to be normally distributed, and sample values (dependent variable) should have ordinality scaled characteristics. By using box plots, we observed that the results of TTCR impact (time to complete review), shared knowledge impact (number of reviewed LOCs), and number of review comments impact are not normally distributed. Since we are comparing the two independent categories, PRs with a specific type of smell (smelly PRs) vs. PRs without that smell (non-smelly PRs), the Mann-Whitney U test is the best candidate for testing the results of these three impacts statistically. Mann-Whitney U test is used to compare whether there is a difference in the dependent variable for two independent groups. For instance, to compare time to complete review (TTCR) in PRs with and without sleeping review smells, TTCR is the dependent variable (which depends on the PR being smelly or not), and independent groups are PRs with sleeping review smell and PRs without sleeping review smell. In summary, (1) TTCR, (2) reviewed lines of code, and (3) number of review comments are dependent variables in mentioned impacts, respectively, and independent groups are smelly PRs and non-smelly PRs. Hypothetically, smelly PRs are more likely to have higher time to complete review. Also, LOCs reviewed by review buddies are more likely to be higher than the correspondent value of non-review buddies. In addition, the number of review comments given by reviewers of smelly PRs differs from the number of review comments in non-smelly PRs. We apply a rank-sum test to measure the statistical significance of these results.

Null Hypothesis 2 *The sum of rankings of the dependent variable (TTCR, reviewed lines of code, or the number of review comments) in smelly and non-smelly groups does not differ.*

Mann-Whitney U test calculates the rank sum of the dependent values in two independent samples (T_1 and T_2). From these values U_1 , U_2 and U (equation 5.2), μ_U and σ_U (expected value of U and standard error of U by equation 5.3) values are calculated to obtain Z and r values (equation 5.4). P-value is calculated from z value. n_1 and n_2 are the sample size of each independent group. In this test, the significance level is considered 0.05. If the calculated p-value is less than the usually agreed α risk of 5 percent (0.05) with a level of 95% confidence, the null hypothesis can be rejected, and it can be concluded that the dependent values differ in two independent groups in each case.

$$U_i = n_1.n_2 + \frac{n_i.(n_i + 1)}{2} - T_i \quad U = \min(U_1, U_2) \quad (5.2)$$

$$\mu_U = \frac{n_1.n_2}{2} \quad \sigma_U = \sqrt{\frac{n_1.n_2.(n_1 + n_2 + 1)}{12}} \quad (5.3)$$

$$Z = \frac{U - \mu_U}{\sigma_U} \quad r = \frac{|z|}{\sqrt{n}} \quad (5.4)$$

Equation 5.4 illustrates the formula to calculate the effect size (r) of smelly PRs on the TTCR, shared knowledge, and the number of review comments impacts on the code review process. Effect size is a value measuring the strength of the relationship between two variables. In general, one can say about the effect strength:

Effect size r less than 0.3 → small effect

Effect size r between 0.3 and 0.5 → medium effect

Effect size r greater than 0.5 → large effect

In Table 5.9 p-value and effect size (r) of Mann-Whitney U test for TTCR of smelly PRs vs. non-smelly PRs are given. Furthermore, Table 5.8 provides the p-value and effect size of the Mann-Whitney U test for the number of LOCs reviewed

by reviewers of high frequent pairs (review buddies) vs. low frequent pairs (non-review buddies). Moreover, Table 5.10 display the p-value and effect size (r) of the Mann-Whitney U test for the number of review comments of smelly PRs vs. non-smelly PRs. In Tables 5.9, 5.8 and 5.10, if the p-value is less than 0.05, the null hypothesis is rejected; hence smelly PRs and non-smelly PRs differ in terms of TTCR or LOCs (workload) or the number of review comments respectively. We used red color in each cell of the tables to show unexpected results where two independent categories do not have a statistically significant difference. In Table 5.9 obtained p-values are less than the significance level. This indicates that the time to complete review of smelly PRs is significantly different from the TTCR of non-smelly PRs.

Main deduction: Distribution of time to complete review in smelly PRs is significantly different from non-smelly PRs. Sleeping review, bypassed negative reviewer, ping pong, and large changeset smell increase the time to complete review. On the other hand, existence of missing context smell in a PR will likely decrease the time to complete a PR review.

P-values in Table 5.8 are less than 0.05; thus review workload of review buddies is significantly different from the review workload of non-review buddies.

Main deduction: Distribution of review workload of review buddies is significantly different from non-review buddies. Pairs with higher occurrence frequencies (review buddies) have higher reviewed LOCs, meaning that review buddies have a heavier review workload; thus, more burden is on review buddies. Even though the average and median of LOCs reviewed by review buddies are higher than normal reviewers and the workload distribution (CBCS score) has a wider range in review buddies, this cannot confirm the code review quality. This means that when review buddies smell occurs in a code review process, the review workload is distributed unevenly across contributors, and more burden is put on reviewers who are review buddies.

In Table 5.10, except for bypassed negative reviewer smell in LibreOffice

Table 5.8: P-value and effect size (r) of statistical test with Mann-Whitney U testing for shared knowledge impact, workload of review buddies and non-review buddies in four Gerrit projects. Expected Results (Blue Color)

Projects	p-value	Effect size (r value)
Qt	1.39e-146	0.44
Eclipse	6.69e-60	0.45
Wireshark	1.34e-11	0.35
LibreOffice	6.16e-22	0.40

projects, the number of review comments in smelly PRs differ from non-smelly PRs in all projects.

Main deduction: Except for bypassed negative reviewer smell in LibreOffice, distribution of number of review comments in smelly PRs is significantly different from non-smelly PRs in all projects. A PR being smelly affects the number of review comments given in the code review of the PR.

Table 5.9: P-value and effect size (r) of statistical test with Mann-Whitney U testing for TTCR of smelly and non smelly PRs in four Gerrit projects. Expected Results (Blue Color)

Smells	Qt		Eclipse		Wireshark		LibreOffice	
	p	r	p	r	p	r	p	r
PRs with sleeping review smell VS PRs without sleeping review smell	0.0	0.86	0.0	0.79	0.0	0.60	0.0	0.72
PRs with bypassed negative re- viewer smell VS PRs without bypassed negative re- viewer smell	4.00e-11	0.02	4.39e-18	0.03	8.53e-17	0.06	3.88e-05	0.01
PRs with ping pong smell VS PRs without ping pong smell	0.0	0.25	0.0	0.23	0.0	0.31	1.54e-227	0.13
PRs with large changeset smell VS PRs without large changeset smell	0.0	0.18	0.0	0.15	3.00e-12	0.05	1.03e-70	0.07
PRs with missing context smell VS PRs without missing context smell	5.01e-246	0.11	0.0	0.17	1.61e-08	0.04	2.96e-276	0.14

Table 5.10: P-value and effect size (r) of statistical test with Mann-Whitney U testing for number of review comments of smelly and non smelly PRs in four Gerrit projects. Expected Results (Blue Color), Unexpected Results (Red Color)

Smells	Qt		Eclipse		Wireshark		LibreOffice	
	p	r	p	r	p	r	p	r
PRs with sleeping review smell VS PRs without sleeping review smell	0.0	0.37	1.10e-117	0.29	9.11e-93	0.43	2.14e-74	0.302
PRs with bypassed negative re- viewer smell VS PRs without bypassed negative re- viewer smell	4.29e-06	0.02	0.03	0.024	0.058	0.04	0.45	0.01
PRs with ping pong smell VS PRs without ping pong smell	0.0	0.51	0.0	0.42	2.47e-97	0.44	6.21e-95	0.34
PRs with large changeset smell VS PRs without large changeset smell	9.57e-194	0.16	1.54e-134	0.25	3.48e-49	0.31	3.84e-12	0.1
PRs with missing context smell VS PRs without missing context smell	2.10e-88	0.11	9.59e-44	0.14	1.25e-09	0.12	6.96e-12	0.11
PRs with review buddies smell VS PRs without review buddies smell	0.0	0.35	7.75e-191	0.309	1.16e-86	0.42	3.72e-99	0.35

5.3 Correlation of A PR Being Smelly VS. Reopened Issues Impact and Abandoned Pull Requests Impact

Chi-square test inquires the relationship between a PR being smelly and the likelihood of reopened issues or abandoned pull requests impacts. The chi-square test of independence is a statistical hypothesis test used to establish the likelihood of a relationship between two nominal or categorical variables when the data is the frequency of values for two categorical variables. For instance, if the data is the frequency of PRs with a specific type of smell linked to reopened issues and the frequency of these smelly PRs linked to non-reopened issues, we can apply chi-square test. The data consists of the results related to several types of smell; by applying the chi-square test, we can compare the relationship between a PR being smelly (of any type) and being linked to reopened issues. We took several steps to apply the chi-square test to the values in Table 5.11. First, the null hypothesis should be determined.

Null Hypothesis 3 *For a PR, being smelly and being linked to reopened issues are independent.*

After defining null hypothesis, also expected value of each case should be calculated. Next, test statistic, χ^2 , is calculated based on observed (O_k) and expected (E_k) values as follow:

$$\tilde{\chi}^2 = \sum_{k=1}^n \frac{(O_k - E_k)^2}{E_k}$$

If the null hypothesis is true, the observed and expected frequencies will be close in value, and the χ^2 statistic will be close to zero.

Table 5.11: An example of data used in chi-square test to analyze the significance of relationship between a PR being smelly and a PR being linked to reopen issues resulted (observed) in Eclipse project

Smells	Linked to reopen issues	Linked to non-reopen issues
PRs with bypassed negative reviewer smell	2	109
PRs with review buddies smell	47	716
PRs with LOCR smell	1512	26355
PRs with ping pong review smell	156	3416
PRs with sleeping review smell	1120	2004
PRs with LCHS smell	691	1315

Table 5.12: χ^2 and p-values of chi-square test for reopened issues impact in four Gerrit projects (df = 5). Expected Results (Blue Color), Unexpected Results (Red Color)

Projects	χ^2	p-value
Qt	1441.80	1.19e-309
Eclipse	4475.10	0.0
Wireshark	514.03	7.44e-109
LibreOffice	1901.80	0.0

Table 5.13: χ^2 and p-values of chi-square test for abandoned PRs impact in four Gerrit projects (df = 3). Expected Results (Blue Color), Unexpected Results (Red Color)

Projects	χ^2	p-value
Qt	2857.99	0.0
Eclipse	990.82	1.76e-214
Wireshark	220.109	1.9e-47
LibreOffice	492.67	1.84e-106

Based on the observed values from the Eclipse project illustrated in Table 5.11 and χ^2 is equal to 4475.10, degree of freedom is 5. To interpret these test statistics in 5% significance level ($\alpha = 0.05$), we check the chi-Squared distribution table within the degree of freedom $DF = 5$, which is 11.070. This means that within $DF = 5$, the statistic value (11.070) has a probability or p-value of 0.05, which is 5%. So, any value less than 11.070 will have a probability of more than 5%. Consequently, we fail to reject the null hypothesis. Therefore, in the Eclipse project, for a PR, being smelly and being linked to reopened issues are not independent. We performed this test in four projects for reopened issues impact and abandoned PRs impact. Tables 5.12 and 5.13 illustrate the χ^2 and p-value of chi-square test for mentioned impacts in four projects. Tables 5.12 shows that the test statistics for all projects reject the null hypothesis and confirm a significant dependency between being smelly and the existence of reopened issues. Furthermore, Table 5.13 illustrates that in all four projects, p-values are less than 0.05, thus being smelly and abandoned PRs are significantly dependent.

5.4 Final Deduction

Figure 5.1: Correlations between code review process smells and their impacts (solid blue lines :confirmed by statistical tests, dashed red lines: rejected by statistical tests)

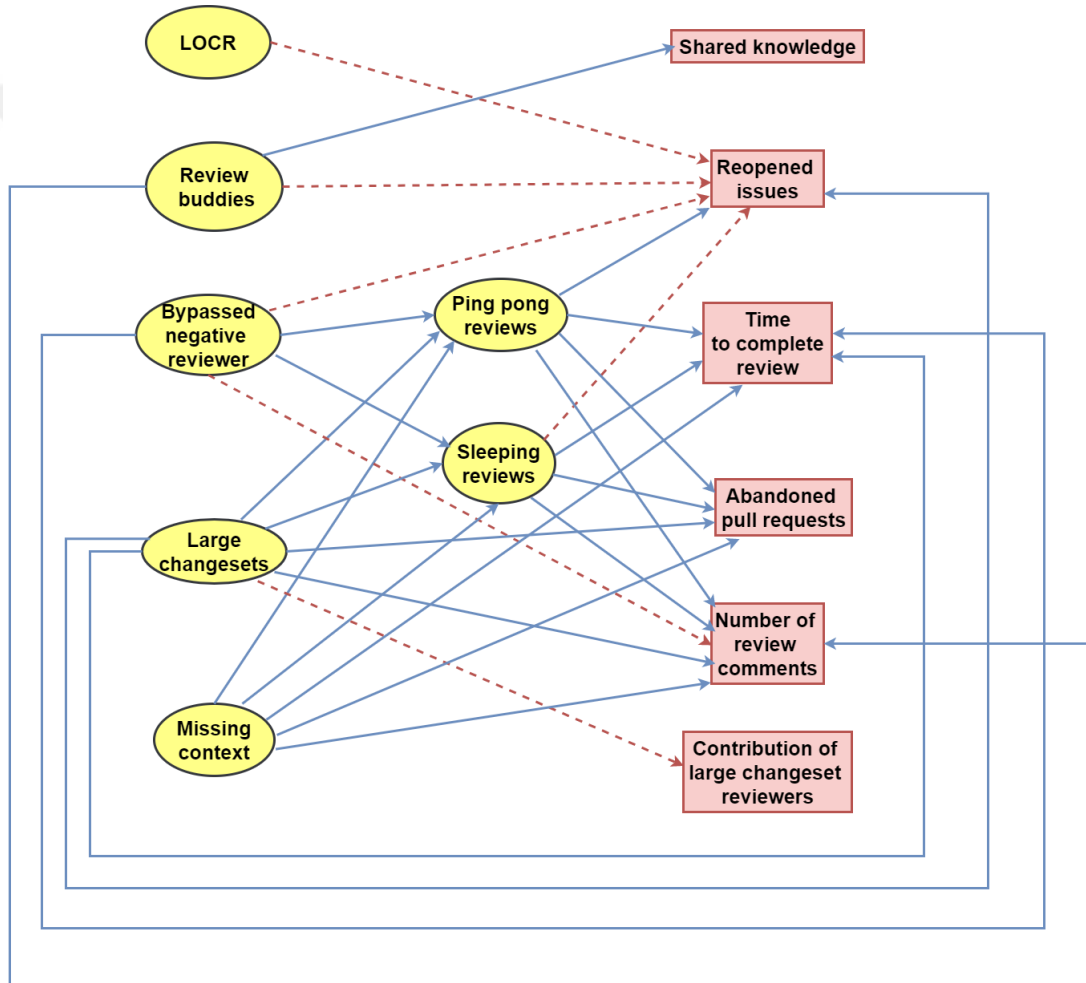


Figure 5.1 illustrates the significant correlations and impacts among code review process smells based on the statistical tests. In Figure 5.1, for each arrow between two smells or a smell and impact, if three out of four projects that we used have statistically significant results, arrow is displayed as solid blue arrow. On the other hand, if less than three projects have significant results, we used dashed red arrow. Blue solid arrows between two particular smells indicate a correlation

between them. On the other hand, dashed red arrows between smells show co-occurrences of two smells that statistical tests do not confirm their correlation. Solid blue arrows between a particular smell type and an impact depict that the smell significantly creates a negative impact based on statistical tests. However, red dashed lines between smells and impacts show that we cannot confirm a relationship between associated smell and impact, and the smell is not the potential reason for the impact.



Chapter 6

Discussion

In this section, we discuss the RQs defined in Section 1 and the implications of our study.

6.1 Answers to Research Questions

In this study, we intend to analyze the impacts and correlations between committers' side and reviewers' side code review process smells. We investigated two research questions about correlations between code review process smells and how to identify impacts of these smells.

RQ1- What are the correlations between the code review process smells?

To confirm the correlations between code review process smells, we categorized code review process smells as committers' side and reviewers' side smells and we defined positive and negative cases and detected PRs with all cases (double-positive where both smells exist in a PR, double-negative where none of the smells are observed in a PR, single-positive and single-negative where only committers'

side smell is observed in PR, single-negative and single-positive where only reviewers side smell is observed in PR). The frequency of PRs in each of these cases enabled us to understand the relationship and significance of the relationship between different smells, how the existence of a committers' side smell leads to the occurrence of a reviewers' side smells and how lack of one smell prevents the emergence of other smells. The occurrence of code review process smells vary depending on several factors. One of the leading causes of process smells might be the differences between projects and their contribution guidelines. For instance, review buddies smell is explicitly mentioned in Qt project contribution guideline. However, other projects do not warn developers about this smell in their contribution guidelines. The next reason is that occurrence of a reviewers' side smell may be triggered by a committers' side smell as bad practices of authors may create possible problems in the review phase. We realized that committers' side smells are highly correlated with reviewers' side smells, especially in the projects where their contribution guidelines lack the proper rules to prevent associated smells. For example, contribution guideline of LibreOffice project warn developers about merging pull requests with a negative vote clearly, yet we observed instances of bypassed negative reviewer smell and correlation of this smell with reviewers' side smells in LibreOffice project. To confirm the correlation between smells, we applied statistical tests to co-occurrence of smells.

RQ2- What are the impacts of code review process smells, and how prevalent are these impacts in OSS projects?

The existence of a code review process smells creates consequences at different levels of the project. To answer the second research question, we studied the characteristics and benefits of the ideal code review process and created a list of impacts based on these characteristics and benefits in code review process in existence of smells. Our goal was to investigate if the code review process smells emerge in a project, how the benefits of ideal code review process endanger. We designed different algorithms to detect possible impacts. We realized that review buddies smell disturbs the balance of shared knowledge among development team, as more workload is put on high frequent pairs who are considered review buddies. To illustrate the differences between workload distribution of review buddies and non-review buddies, we computed code base coverage score of each

reviewer in a pair which is an indicator for amount of work that a reviewer has done for an author. We observed that workload distribution is very different in review buddy pairs and non-review buddy pairs. Furthermore, we analyzed the ratio of reopened issues linked to PRs with a particular smell type and we observed ping pong and large changeset smells increases the ratio of reopened issues. Moreover, by calculation the time to complete review as the duration from when first changeset is submitted till the last non-bot comment given by a reviewer, we confirm that existence of code review process smells increases the time to complete review. In addition, by computing number of review comments as sum of inline and file-level comments and compare distribution of review comments in smelly and non-smell PRs, we realized that existence of code review process smells changes the number of review comments. Also, code review process smells increases the number of abandoned PRs. We also observed that reviewing a large changeset has not an impact on reviewers' contribution in upcoming review tasks.

6.2 Implications to Practitioners and Researchers

Researcher and practitioner's ability to generalize, discuss and apply results are made possible by classifications [28]. In this study, we intended to categorize and analyze the consequences of committers' side, and reviewers' side code review process smells. Knowing the consequences and impacts of bad practices that committers and reviewers perpetrate can help developers of a project to know the outcomes of their nonconformance. Impacts like shared knowledge, reopened issues, and abandoned pull requests increase project's maintenance costs. Unbalance shared knowledge makes review buddies reviewers indispensable if they leave the project and increase the project's bus factor. In addition, reopened issues and abandoned PRs increase the maintenance costs and create technical debt in the project if a release happens despite these problematic pull requests and issues. Knowing these smells and impacts benefit code review and issue tracking systems to enhance performance. A code review tool could be enhanced by categorizing

smells as committers' or reviewers' side smells and knowing the initial point of these smells in the lifecycle of a pull request. For example, to prevent bypassed negative reviewer smell, the code review tool could be enhanced by providing a warning if a developer tries to merge a PR with such a smell (unchanged negative vote) or even forbid the merger. Warning committers could handle missing context smell as well in submitting phase if the commit message is too short or does not contain the issue number. Large changeset smell also could be managed by warning large changeset committers in submitting phase. Lack of code review and sleeping review smells should also be handled with better assignment tools to assign proper reviewers. Reviewers of a PR also could get notified after two days to prevent impacts of sleeping review smell in the project. In addition, code reviewing tools could be enhanced to handle review buddies smell as it provides a warning if the committer assigns a review to pre-determined "buddies" based on each developer's network in the project.

Chapter 7

Threats to Validity

Most of our concerns regarding the internal validity of this study come from the thresholds we assume during our analysis. We have tried to keep these values reasonable by either considering software development best practices or the general properties of the data we obtained. For example, while setting the threshold for high & low-frequency pairs for review buddies, we used the underlying statistical properties of the data. Another concern regarding internal validity is due to hard-to-detect causal relations with the smells and impacts: even though impacts and smells in this study are correlated (we tested the co-occurrences with statistical tests to confirm correlations), we cannot guarantee causal relationships between them because proving such relationships is not straightforward. Proving causality would require one to clear the smells from a project's actual code review workflow and then observe whether the impacts diminish. Such an idea cannot be applied to a real-world project and is out of the scope of this study. How Gerrit PRs are structured and applied is also another point of concern. Unlike PRs in traditional Git platforms like GitHub, which allow creating branches with multiple commits and the cumulative changes made by these commits to be applied back to the main or stable branch, each PR in Gerrit is one commit. If any amends or updates need to be made to the PR after submitting its first commit, an entirely new commit is made and linked with the PR. If a PR is accepted, then the commit that is applied to the target branch is the one that was last submitted

as part of the PR. The previous commits linked with that PR have no bearing on the target. This is why for smells that require commit-specific information, such as lines of code and checking if a PR has a bypassed negative reviewer smell, we have limited our analysis to the last commit in a given PR.

Another threat to internal validity is manually extracting non-human contributors' names, such as bot reviews. Since non-human contributors to each project were inspected manually some cases might have been skipped.

Another threat to the validity of our work is regarding the review buddies smell detection algorithm. Based on the definition, if an author-reviewer pair's occurrence frequency is more than the defined threshold, the reviewer is the review buddy of the author. One of the concerns is related to the case when a reviewer is a review buddy of several authors. Each of these authors can create a pair with this reviewer. This case may happen when a reviewer has proper knowledge about a part of the codebase and authors of changesets that trigger this part of codebase, assign the same person over and over to get their PRs reviewed.

Another concern of our research is related to review buddies smell and author reviewer pairs. Even though we compute the occurrence frequency of pairs in the evolution history of the projects, we do not consider time as a factor in our algorithm which means that the occurrence frequency of each pair was equal to one once author and reviewer of the associated pair had accompanied for the first time. This means that each high frequent pair (review buddies) was a low frequent pair with occurrence frequency less than the threshold for a while in the evolution history, but since we did not consider past collaboration network of developers and high or low frequent label of a pair does not change over time based on the contribution of developers' of associated pair.

We could not observe a particular pattern related to the contribution of large changeset reviewers impact. Threats to construct validity involve how the theory and the observation relate, and in our investigation, these threats are mostly caused by the measurements we took about how we evaluated the increase in large changeset reviewer's expertise during a code review with CBCS. Since expertise

and tacit knowledge are very hard to measure quantitatively, more investigation and analysis of this impact are needed.

Our last concern of internal validity is related to statistical tests that we performed on our empirical results to analyze the significance of our results. During statistical analysis with Fisher's exact test, even though we reported odds ratio values in each project, we only considered the p-value as a factor to confirm our results of correlations between smells, reopened issues impact, and abandoned pull requests impact. The odds ratio is important to analyze the likelihood of observing negative cases in projects, and if obtained odds ratio is less than 1, the likelihood of observing negative cases is more than positive cases. However, we considered the p-value a decision-maker factor to analyze the significance of our results.

The concerns regarding the external validity of this project are mostly based on the generalizability of our smell and impact detection methods. This study uses only four OSS projects to analyze correlations and impacts of code review process smells. We acknowledge that some smells in this study might be undetectable in certain repositories depending on how they store and organize their code review-related data. In such cases, we expect practitioners to draw their conclusions from the example repositories in this study. Another reason for not observing these smells might be the workflow rules blocking out the smells completely. However, this is a positive thing because it shows that the management of the repository acknowledged the potentially problematic practices and prevented them.

Chapter 8

Conclusion and Future Work

In this study, we investigated the correlations and impacts of code review process smells in different aspects. We analyzed six different impacts of code review process smells; (1) shared knowledge, (2) reopened issues, (3) time to complete review, (4) number of review comments, (5) abandoned pull requests and (6) contribution of large changeset reviewers. Subsequently, we quantitatively analyzed these impacts and conducted an empirical analysis by mining four OSS projects (Qt, Eclipse, Wireshark, LibreOffice) on 244,903 pull requests in total.

- Bypassed negative reviewer smell is likely to lead to reviewers' side smells; sleeping reviews, and ping pong reviews. 48-68% of the bypassed reviews take more than two days to get reviewed. Also, in QT and Wireshark projects, more than 80% of bypassed negative reviewer PRs contain more than three review cycles (ping pong smell). The ping pong smell ratio is less for Eclipse and LibreOffice projects, yet the results are statistically significant. Re-assignment of a new reviewer for a positive vote can increase the time and number of iterations in code review.
- More than 25% of the PRs with large changesets suffer from sleeping review smell in all four projects. Moreover, large changeset smell correlates with ping pong smell in all project except LibreOffice project. Obtained results

from LibreOffice project do not confirm a significant correlation between LCHS and ping pong smell in this project.

- Missing context smell is correlated with ping pong smell and sleeping review smell.
- Review buddies smell ratio is between 3-5% in all four projects, yet the sum of LOCs reviewed by reviewers of high frequent pairs is significantly different from the sum of LOCs in low frequent pairs. This result confirms that when review buddies smell occurs in a project, most of the workload is put on the review buddies.
- Issues linked to PRs with ping pong smell and large changeset smell are significantly more likely to reopen than issues linked to PRs with other smells.
- Time to complete review (TTCR) of (1) sleeping review, (2) bypassed negative reviewer, (3) ping pong, (4) large changeset, and (5) missing context smells are significantly different from TTCR of normal and proper PRs that have not correspondent smells. Except for missing context smell, TTCR or smelly PRs have higher average and median values than TTCR or non-smelly PRs in four projects. A smelly PR can increase the time to complete review of the PR.
- Abandoned pull requests impact is less than 20% in sleeping reviews, less than 4% in ping pong reviews, and less than 21% in PRs with large changesets and missing context smells in all four Gerrit projects, yet a PR getting abandoned is significantly related to PR being smelly. Based on the statistical tests and obtained odds ratios related to this impact, sleeping review smell and abandoned pull request impact has the highest odds ratio compared to other smells.
- Number of review comments in PRs with (1) sleeping review, (2) bypassed negative reviewer, (3) ping pong review, (4) large changesets, (5) missing context, and (6) review buddies smells are significantly different from the number of review comments in non-smelly PRs. Code review process smells can affect the number of comments reviewers obtain as feedback.

- We cannot observe a positive impact of reviewing large changesets on reviewers' future contributions. The contribution pattern of reviewers does not follow a repetitive pattern.

In future work, we will continue analyzing correlations and impacts of code review process smells in other projects using code review tools such as GitHub to generalize our results. Moreover, to investigate code review process smells even after product release, we will analyze the technical debt that these smells might create in the project. Furthermore, some approaches to prevent these smells and to avoid their impacts during the software development process would be suggested by comparing the behavior of the developers in two different projects; (1) project with a minimum percentage of a particular smell and (2) project with the maximum percentage of a particular smell. Comparing these two projects and observing developers' behavior in these projects may reveal some techniques to prevent code review process smells throughout the software development process.

Bibliography

- [1] E. Doğan and E. Tüzün, “Towards a taxonomy of code review smells,” *Information and Software Technology*, vol. 142, p. 106737, 2022.
- [2] Z. Soh, A. Yamashita, F. Khomh, and Y.-G. Guéhéneuc, “Do code smells impact the effort of different maintenance programming activities?,” in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, pp. 393–402, 2016.
- [3] I. Ahmed, C. Brindescu, U. A. Mannan, C. Jensen, and A. Sarma, “An empirical examination of the relationship between code smells and merge conflicts.,” in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* (A. Bener, B. Turhan, and S. Biffl, eds.), pp. 58–67, IEEE Computer Society, 2017.
- [4] R. Verdecchia, R. A. Saez, G. Procaccianti, and P. Lago, “Empirical evaluation of the energy impact of refactoring code smells.,” in *ICT4S*, pp. 365–383, 2018.
- [5] A. Ram, A. A. Sawant, M. Castelluccio, and A. Bacchelli, “What makes a code change easier to review: an empirical investigation on code change reviewability,” in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 201–212, 2018.
- [6] O. Baysal, O. Kononenko, R. Holmes, and M. W. Godfrey, “The influence of non-technical factors on code review,” in *20th Working Conference on Reverse Engineering*, WCRE’13, pp. 122–131, Oct. 2013.

- [7] M. Chouchen, A. Ouni, R. G. Kula, D. Wang, P. Thongtanunam, M. W. Mkaouer, and K. Matsumoto, “Anti-patterns in modern code review: Symptoms and prevalence,” in *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 531–535, 2021.
- [8] L. Pascarella, D. Spadini, F. Palomba, and A. Bacchelli, “On the effect of code review on code smells.,” *CoRR*, vol. abs/1912.10098, 2019.
- [9] P. C. Rigby and M.-A. Storey, “Understanding broadcast based peer review on open source software projects,” in *2011 33rd International Conference on Software Engineering (ICSE)*, pp. 541–550, IEEE, 2011.
- [10] T. Hirao, S. McIntosh, A. Ihara, and K. Matsumoto, “Code reviews with divergent review scores: An empirical study of the openstack and qt communities,” *IEEE Transactions on Software Engineering*, vol. 48, no. 1, pp. 69–81, 2022.
- [11] T. Hirao, A. Ihara, Y. Ueda, P. Phannachitta, and K.-i. Matsumoto, “The impact of a low level of agreement among reviewers in a code review process,” in *IFIP International Conference on Open Source Systems*, pp. 97–110, Springer, 2016.
- [12] A. Bacchelli and C. Bird, “Expectations, outcomes, and challenges of modern code review,” in *Proceedings of the 2013 International Conference on Software Engineering, ICSE’13*, pp. 712–721, May 2013.
- [13] M. Caulo, B. Lin, G. Bavota, G. Scanniello, and M. Lanza, “Knowledge transfer in modern code review,” in *Proceedings of the 28th International Conference on Program Comprehension*, pp. 230–240, 2020.
- [14] P. C. Rigby and D. M. German, “A preliminary examination of code review processes in open source projects,” tech. rep., Vol. 6. No. 8 Technical Report DCS-305-IR, University of Victoria, 2006.
- [15] N. Zazworka, V. R. Basili, and F. Shull, “Tool supported detection and judgment of nonconformance in process execution.,” in *ESEM*, pp. 312–323, IEEE Computer Society, 2009.

- [16] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, “The impact of code review coverage and code review participation on software quality: a case study of the qt, vtk, and itk projects.,” in *MSR* (P. T. Devanbu, S. Kim, and M. Pinzger, eds.), pp. 192–201, ACM, 2014.
- [17] G. Bavota and B. Russo, “Four eyes are better than two: On the impact of code reviews on software quality,” in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 81–90, 2015.
- [18] S. Khatoonabadi, D. E. Costa, R. Abdalkareem, and E. Shihab, “On wasted contributions: Understanding the dynamics of contributor-abandoned pull requests: A mixed-methods study of 10 large open-source projects,” *ACM Trans. Softw. Eng. Methodol.*, 2022.
- [19] Z. Li, Y. Yu, T. Wang, G. Yin, S. Li, and H. Wang, “Are you still working on this? an empirical study on pull request abandonment,” *IEEE Transactions on Software Engineering*, vol. 48, no. 6, pp. 2173–2188, 2022.
- [20] “Gerrit documentation.” <https://gerrit-review.googlesource.com/Documentation/config-labels.html>. Accessed: 2022-06-11.
- [21] M. E. Fagan, “A history of software inspections.,” in *Software Pioneers* (M. Broy and E. Denert, eds.), pp. 562–573, Springer Berlin Heidelberg, 2002.
- [22] E. Kempin, “Gerrit - concepts and workflows.” online, January 2019. https://docs.google.com/presentation/d/1C73UgQdzZDwOgzpaEqIC6SPujZJhqamyq01X0HjH-uk/view#slide=id.g4d6c16487b_31_60.
- [23] S. Dueñas, V. Cosentino, G. Robles, and J. M. Gonzalez-Barahona, “Perceval: software project data at your will,” in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings*, pp. 1–4, 2018.
- [24] “Review policy - qt wiki.” https://wiki.qt.io/Review_Policy. Accessed: 2022-03-30.

- [25] “Eclipse development process — the eclipse foundation.” https://www.eclipse.org/projects/dev_process/development_process_2010.php. Accessed: 2022-03-30.
- [26] “Development/submittingpatches - the wireshark wiki.” <https://wiki.wireshark.org/Development/SubmittingPatches>. Accessed: 2022-03-30.
- [27] “Development/getinvolved - the document foundation wiki.” <https://wiki.documentfoundation.org/Development/GetInvolved>. Accessed: 2022-03-30.
- [28] I. Vessey, V. Ramesh, and R. L. Glass, “A unified classification system for research in the computing disciplines,” *Information and Software Technology*, vol. 47, no. 4, pp. 245–255, 2005.