Toward the Optimization of Peer Review Effort in DevOps: An Empirical Investigation Using Merge Request Traces

by

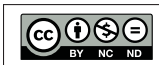
Samah KANSAB

MANUSCRIPT-BASED THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
Ph.D.

MONTREAL, JULY 08, 2026

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC


Samah Kansab, 2026



This Creative Commons license allows readers to download this work and share it with others as long as the author is credited. The content of this work cannot be modified in any way or used commercially.

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Pr. Francis Bordeleau, Thesis supervisor
Department of Software Engineering and IT, École de technologie supérieure

Pr. Erik Andrew Poirier, Chair, Board of Examiners
Department of Construction Engineering, École de technologie supérieure

Pr. Ali Ouni, Member of the Jury
Department of Software Engineering and IT, École de technologie supérieure

Pr. Yann-Gaël Guéhéneuc, External Independent Examiner
Department of Computer Science and Software Engineering, Concordia University

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC

ON JULY 02, 2026

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

FOREWORD

This doctoral thesis is presented in the form of a thesis by compilation. The research conducted during the doctoral program resulted in four conference papers published in peer-reviewed venues, two conference papers currently submitted, three journal papers submitted to peer-reviewed journals, and one additional journal paper in preparation. The content of these papers was adapted where needed to fit the overall structure and continuity of the thesis. Although the integrated chapters remain faithful to the core contributions of the original papers, limited modifications were introduced to ensure consistency of presentation, terminology, and formatting within the École de technologie supérieure thesis template. Hyperlinks to the original published, submitted, or in-preparation papers, together with their associated replication packages when available, are provided in the header of each corresponding chapter and are summarized in Table 1 of the general introduction.

While each paper addresses a particular dimension of the broader research problem, the contributions are strongly interconnected and together form the scientific foundation of this thesis. To provide a unified manuscript, the compiled papers are framed by a general introduction, intermediate chapter transitions where appropriate, and a general discussion and conclusion that synthesize the contributions, discuss their limitations, and identify future research directions.

Generative artificial intelligence tools were used in a limited and assistive capacity during the preparation of this thesis. GPT-5.4 Thinking and Claude Sonnet 4.6 were used to support the revision and improvement of parts of the written text. Claude Opus 4.6 was also used to assist in improving portions of the code included in the replication packages. In all cases, the author carefully reviewed, verified, and validated the resulting text and code before inclusion in the thesis or submission for publication.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor, Professor Francis Bordeleau, for his constant support, trust, and guidance throughout this doctoral journey. More than simply supervising this work, he gave me the opportunity to truly make this thesis my own. By allowing me the freedom to explore, to make mistakes, and to learn through experience, he helped me grow not only as a researcher, but also as an independent thinker. His advice, encouragement, and steady presence created the conditions that made this work possible.

I would also like to warmly thank our industrial partners for their support and collaboration throughout this thesis. My sincere thanks go to Dr. Ali Tizghadam from TELUS for his time, valuable advice, continuous involvement and support. I am also grateful to Dr. Bassem Guendy from Kaloom for his unconditional support, generosity, and constant presence throughout this work.

I would also like to thank all the teachers, instructors, professors, and mentors who have contributed to my academic journey since my earliest years of learning. Every stage of education builds upon the previous one, and I remain deeply grateful to everyone who helped shape my path. Without those first lessons, even the most advanced achievements would not have been possible.

My deepest gratitude goes to my parents, Aïcha and Abdelkader, my sisters Linda, Bouchra, and Hayem, and my brother Wassim, for their unwavering love and support, and to my extended family — with a special tribute to my dear departed loved ones, whose memory remains a guiding presence.

I am equally thankful to my husband Reda, who stood by me through the most difficult moments while pursuing his own Ph.D., and to my parents-in-law Malika and Hamid, and my family-in-law, for welcoming me as a second family.

Finally, I dedicate this thesis to all my friends in Algeria, France, and Canada, and to the colleagues with whom I shared this journey.

Optimisation de l'effort de revue par les pairs dans les environnements DevOps : une étude empirique fondée sur les traces de merge requests

Samah KANSAB

RÉSUMÉ

La revue par les pairs constitue une étape centrale des flux de travail DevOps modernes, où les changements logiciels sont examinés avant leur intégration au moyen de mécanismes tels que les merge requests (MRs). Bien que les travaux antérieurs aient étudié la revue par les pairs sous plusieurs angles, l'effort de revue est encore souvent approximé par des mesures temporelles qui confondent le travail de revue proprement dit avec les temps d'attente, la disponibilité des réviseurs et les contraintes organisationnelles. Cette thèse étudie l'effort de revue par les pairs comme un phénomène de processus logiciel plutôt que comme un simple indicateur temporel, et examine comment il peut être mieux caractérisé, analysé et optimisé dans les environnements DevOps.

La thèse adopte une approche empirique fondée sur des traces de MRs recueillies auprès de deux partenaires industriels (Kaloom et TELUS) ainsi que dans des projets open source GitLab. Elle établit les traces de MRs comme un instrument d'observabilité qui capture non seulement l'activité de revue, mais aussi les perturbations environnementales, les transitions technologiques, les priorités organisationnelles et les interactions entre automatisation et humains. Elle introduit la quantité de changements réalisés après soumission comme un indicateur d'effort indépendant du temps, montre qu'il capture une dimension distincte du travail induit par la revue et le prédit dès la création de la MR avec une AUC atteignant 0,88. Elle examine comment l'effort et la qualité de la revue varient selon les types de MRs (développement, configuration, documentation), en mettant en évidence le profil particulier des changements de configuration et l'effet amplificateur de la présence de bogues. Elle définit une taxonomie des déviations de flux de travail, montre qu'elles touchent jusqu'à 37% des MRs industrielles, les détecte avec une précision atteignant 91% et démontre leur impact sur l'analyse de l'effort. Elle étend ensuite l'analyse en amont, en reliant l'effort de revue aux pratiques d'estimation des tâches, puis en aval, en reliant les décisions de revue aux résultats CI/CD et en prédisant les MRs pouvant être auto-approuvées de manière sûre avec une AUC supérieure à 0,82. Enfin, elle présente RevMine, un outil assisté par grand modèle de langage pour la collecte et l'analyse reproductibles des données du pipeline DevOps sur GitHub et GitLab, intégrant les données de planification de type Kanban, les données de revue par les pairs et les données de livraison CI/CD dans un même flux d'analyse transversal, et opérationnalisant ainsi la chaîne complète amont–revue–aval étudiée dans les chapitres précédents.

À travers ces contributions, la thèse soutient que l'effort de revue par les pairs est un construit multi-dimensionnel qu'aucun indicateur unique ne peut capturer adéquatement, et que son optimisation est un problème de calibration plutôt que de minimisation : l'effort doit être réduit là où il s'avère non nécessaire et préservé là où il est sous-investi. En reliant la planification en amont, l'étape de revue elle-même et les résultats de livraison en aval, la thèse fournit des

fondements empiriques, des artefacts méthodologiques et un support outillé pour interpréter et gérer l'effort de revue par les pairs dans les contextes DevOps.

Mots-clés: Revue par les pairs, Merge requests, Effort de revue, DevOps, Amélioration des processus logiciels, Génie logiciel empirique

Toward the Optimization of Peer Review Effort in DevOps: An Empirical Investigation Using Merge Request Traces

Samah KANSAB

ABSTRACT

Peer review is a central stage of modern DevOps workflows, where software changes are examined before integration through mechanisms such as merge requests (MRs). Although prior research has studied peer review from several angles, review effort is still commonly approximated by temporal measures that conflate review work with waiting time, reviewer availability, and organizational urgency. This thesis studies peer review effort as a software process phenomenon rather than a purely temporal indicator, and investigates how it can be better characterized, analyzed, and optimized in DevOps environments.

The thesis follows an empirical approach based on MR traces from two industrial partners (Kaloom and TELUS) and open-source GitLab projects. It establishes MR traces as an observability instrument that captures review activity together with environmental disruption, technological transitions, organizational priorities, and automation–human interactions. It introduces the amount of post-submission change as a time-independent effort indicator, shows that it captures a distinct dimension of review work, and predicts it at MR creation time with an AUC of up to 0.88. It examines how review effort and quality vary across development, configuration, and documentation MRs, highlighting the specific profile of configuration changes and the amplifying effect of bug presence. It defines a taxonomy of workflow deviations, shows that they affect up to 37% of industrial MRs, detects them with up to 91% accuracy, and demonstrates their impact on effort analysis. It then extends the analysis upstream, by linking review effort to task estimation, and downstream, by relating review decisions to CI/CD outcomes and predicting safely self-approvable MRs with an AUC above 0.82. Finally, it presents RevMine, an LLM-assisted tool for reproducible DevOps-pipeline data collection and analysis across GitHub and GitLab, which integrates Kanban-style planning data, peer review data, and CI/CD delivery data into a single cross-stage workflow and thereby operationalizes the full upstream–review–downstream chain studied in the preceding chapters.

Across these contributions, the thesis argues that peer review effort is a multi-dimensional construct that no single indicator captures adequately, and that its optimization is a calibration problem rather than a minimization one: effort should be reduced where evidence shows it is unnecessary and preserved where it is under-invested. By connecting upstream planning, the review stage itself, and downstream delivery outcomes, the thesis provides empirical foundations, methodological artifacts, and tooling support to interpret and manage peer review effort in DevOps settings.

Keywords: Peer review, Merge requests, Review effort, DevOps, Software process improvement, Empirical software engineering

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF ABBREVIATIONS

ACC	Accuracy
AI	Artificial Intelligence
ALE	Accumulated Local Effects
ANOVA	Analysis of Variance
API	Application Programming Interface
AUC	Area Under the ROC Curve
BERT	Bidirectional Encoder Representations from Transformers
BOCA	Build Or Configuration Adjustments (deviation category)
BS	Brier Score
CC	Code Cleaning (deviation category)
CD	Continuous Delivery
CI	Continuous Integration
CI/CD	Continuous Integration / Continuous Delivery
CP	Control Plane (Kaloom project group)
CPC	Cross-Platform Consistency
DevOps	Development and Operations
DP	Data Plane (Kaloom project group)
ECS	Empty Change Set (deviation category)
EOW	Experimental Or Work-in-progress (deviation category)

ETS	École de technologie supérieure
FAR	Field Accuracy Rate
FCR	Field Completeness Rate
FDR	Feature Dependency Recall
FDSR	Full Dependency Satisfaction Rate
FM	F-measure
FPGA	Field-Programmable Gate Array (Kaloom project group)
GH	GitHub
GL	GitLab
HC	Huge Changes (deviation category)
IaC	Infrastructure as Code
JSON	JavaScript Object Notation
KNN	K-Nearest Neighbors
LLM	Large Language Model
LT	Lead Time
LU	Library Update (deviation category)
MAE	Mean Absolute Error
MCC	Matthews Correlation Coefficient
ML	Machine Learning
MP	Management Plane (Kaloom project group)

MR	Merge Request
MSE	Mean Squared Error
MSR	Mining Software Repositories
MV	Multitasking Variable
NLP	Natural Language Processing
OR	Odds Ratio
ORH	Out of Regular Hours
OSS	Open-Source Software
PF	Platform (Kaloom project group)
PR	Pull Request
PRC	Precision
RC	Revert Commits (deviation category)
RCL	Recall
REST	Representational State Transfer
RLT	Reference Lead Time
RMSE	Root Mean Squared Error
ROC	Receiver Operating Characteristic
RQ	Research Question
SDN	Software-Defined Networking
SLE	Service Level Expectation

SMOTE	Synthetic Minority Over-sampling Technique
SPI	Software Process Improvement
TF-IDF	Term Frequency–Inverse Document Frequency
TINAA	TELUS Intelligent Network Analytics and Automation
TWC	Total Waiting-Column time
WIP	Work In Progress
XML	eXtensible Markup Language
YAML	YAML Ain’t Markup Language

LIST OF SYMBOLS AND UNITS OF MEASUREMENTS

i	Generic index for an issue, task, or MR
d	Developer (contributor)
t	Time instant
k	Workflow column or status label
s	Estimated task size (T-shirt size mapped to a Fibonacci weight) or sample size (context-dependent)
ss	Finite-population-corrected sample size
z	Z-score for a given confidence level
p	Estimated population proportion
c	Margin of error
P	Total population size
α	Statistical significance level
τ	Tolerance band around the median reference lead time
$w(i)$	Fibonacci-mapped weight of task i
$\mathcal{W}$	Set of waiting workflow labels
$\mathcal{A}_d(t)$	Set of active tasks assigned to developer d at time t
$\mathcal{T}_i$	Time interval during which task i is active
$\mathbf{1}[\cdot]$	Indicator function
$Q_q(\cdot)$	Empirical q-quantile
$\mathsf{Has}(i, k)$	Predicate: true if task i ever carries label k

$\mathrm{dur}_i(k)$	Duration that task i spent in column k
LT_i	Lead time of issue i (from *In Progress* to *Done*/*Closed*)
RLT_s^{SLE}	Service-level-expectation reference lead time at size s (85th percentile of LT)
RLT_s^{MED}	Median reference lead time at size s
$\mathrm{next}(s)$	Immediate successor of size class s
$\mathrm{WaitExp}_i$	Binary indicator that task i ever entered a waiting state
TWC_i	Total time task i spent in waiting columns
$\mathrm{WaitingPct}_i$	Share of lead time that task i spent waiting
$\mathrm{ExecPct}_i(k)$	Share of lead time spent by task i in execution column k
U_i	Indicator that task i was underestimated
$\mathrm{WL}_d(t)$	Workload of developer d at time t
MV_i	Multitasking variable for task i
r_{rb}	Rank-biserial correlation effect size
δ	Cliff's δ effect size
V	Cramér's V effect size
κ	Cohen's κ inter-rater agreement
R^2	Coefficient of determination (used in redundancy analysis)
U	Mann–Whitney U test statistic

INTRODUCTION

Context and motivation

Software Process Improvement (SPI) aims to improve how software is planned, developed, validated, and delivered, with goals such as higher software quality, faster and more stable delivery cycles, and more effective use of engineering resources. These goals are increasingly pursued through DevOps, which integrates development and operations and supports continuous delivery through automation and feedback Humble & Farley (2010); Riungu-Kalliosaari, Mäkinen, Lwakatare, Tiihonen & Männistö (2016); Brown, Forsgren, Humble, Kersten & Kim (2016). Although automation plays a central role in DevOps workflows, some stages still rely heavily on human judgment, and peer review is one of the most prominent among them.

Peer review examines a proposed software change before its integration into the main codebase. On platforms such as GitLab, it is commonly conducted through Merge Requests (MRs), in which reviewers inspect changes, provide comments, request revisions, and decide whether the change is ready to be merged. Prior work has shown that peer review contributes to both technical and collaborative outcomes, including improved readability (Bavota & Russo, 2015), reduced defect proneness (McIntosh, Kamei, Adams & Hassan, 2014), and stronger knowledge sharing among developers (Bacchelli & Bird, 2013). Positioned between change submission and integration, peer review also matters for broader SPI objectives such as workflow efficiency and delivery reliability (Badampudi, Unterkalmsteiner & Britto, 2023; Wang & Wu, 2022). Its practical importance is reflected in industrial adoption: a 2022 survey reported that 84% of surveyed technology companies had a defined peer review process in place (TrustRadius, 2022).

Software engineering research has examined peer review from many perspectives, including reviewer assignment, review participation, review-comment usefulness, and timing-related aspects such as time to first comment (Bacchelli & Bird, 2013; Thongtanunam, McIntosh,

Hassan & Iida, 2015, 2017b; Kononenko, Baysal & Godfrey, 2016; Rahman, Stallings & Williams, 2018; Hasan, Macedo, Tian, Adams & Ding, 2023). Among these directions, peer review effort has received increasing attention because review is not cost-free: it consumes developer time, can delay integration, and competes with other activities in fast-paced DevOps settings Olewicki, Habchi & Adams (2024b); Chen, Rigby & Nagappan (2022). Better understanding this effort can support more efficient reviewer allocation, earlier identification of changes likely to require substantial follow-up work, and more informed trade-offs between review thoroughness and delivery constraints Chouchen & Ouni (2024); Yang *et al.* (2025). Existing work has largely operationalized peer review effort through temporal measures such as review duration or completion time and has studied factors influencing this temporal view, including review prioritization and early outcome prediction (Chouchen, Ouni, Olongo & Mkaouer, 2023; Zhang, Yu, Gousios & Rastogi, 2022; Maddila *et al.*, 2023; Huang, Jia, Zhou, Hong & Chen, 2020; Fan, Xia, Lo & Li, 2018; Saini & Britto, 2021; Islam, Ahmed, Shahriyar, Iqbal & Uddin, 2022). While useful, these measures provide only an indirect view of peer review effort.

Problem statement

The main problem addressed by this thesis is that peer review effort in DevOps workflows still lacks established metrics and robust analytical methods beyond time-based indicators to support its effective characterization and optimization. Although peer review itself has been extensively studied, the effort it generates remains poorly understood for the following specific reasons.

First, time reflects the pace of peer review, but not the work generated by review itself. Review completion time is shaped not only by corrective activity, but also by reviewer availability, prioritization, waiting periods, and organizational context (Olewicki *et al.*, 2024b; Yu, Wang, Filkov, Devanbu & Vasilescu, 2015). In industrial settings, deadlines, workload, and informal communication outside the review tool may accelerate completion without reducing actual effort (Maddila *et al.*, 2023; Ebert, Castor, Novielli & Serebrenik, 2021). In open-source

settings, timelines may likewise diverge from revision effort because of reviewer availability, workload, and project-specific priorities (Yu *et al.*, 2015; Zhang *et al.*, 2022; Khatoonabadi, Abdellatif, Costa & Shihab, 2024). Time-based indicators therefore provide only a partial and potentially confounded view of peer review effort across development contexts (Zhang *et al.*, 2022; Kononenko *et al.*, 2016; Pascarella, Spadini, Palomba, Bruntink & Bacchelli, 2018).

Although no single universally accepted metric exists for peer review effort, a more direct perspective can be obtained by examining the corrective work performed after initial submission. In response to review requests, authors may revise implementation, configuration, or documentation before integration (Fregnan, Petrulio & Bacchelli, 2022). While such post-submission changes do not capture peer review effort exhaustively, they provide an observable manifestation of review-related work and a useful complement to time-based indicators. Prior work also suggests that review practices vary across artifact types, meaning that peer review should not be analyzed uniformly across all MRs (Nejati, Alfadel & McIntosh, 2023; Spadini, Aniche, Storey, Bruntink & Bacchelli, 2018).

A related challenge is that not all MRs follow the same review workflow. In industrial repositories, some MRs are created for urgent reversions, library updates, temporary coordination purposes, or incomplete work. Such cases do not reflect the standard review–revise–approve–merge process expected for a planned MR in a typical sprint-based software development workflow in the same way as regular MRs. When analyzed together with standard MRs, they may distort effort measures and bias comparisons across MR categories. A rigorous analysis of peer review effort must therefore account for such workflow deviations.

Peer review effort should also not be studied in isolation from the surrounding software process. Upstream planning and task estimation may influence the corrective work later required during review, while downstream delivery outcomes, including CI/CD results, may provide additional evidence about the adequacy of review decisions. Studying peer review effort in relation to

these preceding and following activities supports a broader SPI perspective and enables a more process-oriented understanding of review work in DevOps environments.

Together, these gaps restrict the ability of both practitioners and researchers to understand when peer review effort is necessary, when it becomes excessive, and how it can be managed more effectively to support integration quality and process efficiency.

Objectives

The main objective of this thesis is to contribute to the understanding and optimization of peer review effort by establishing the empirical, methodological, and practical foundations required to analyze it within DevOps workflows.

The specific objectives of this thesis are:

- O1. Establish baselines, metrics, and categorization approaches for analyzing peer review effort beyond time-based indicators.
- O2. Optimize peer review effort by identifying situations in which the level of review effort can be reduced or better adapted to the characteristics and risk of the change.
- O3. Investigate peer review as a workflow stage embedded in the broader DevOps process, including its relationships with upstream planning activities and downstream delivery outcomes.
- O4. Develop reusable methodological and tool support to enable reproducible peer review analytics and facilitate transfer to practice.

Approach and methodology

This thesis follows an empirical, process-oriented approach grounded in peer review traces collected from three complementary contexts: two industrial settings, through our collaborations with **Kaloom** and **TELUS**, and a set of open-source GitLab projects. Together, these contexts

provide a broad empirical basis for studying peer review in environments where software changes are continuously submitted, reviewed, and integrated through Git-based DevOps workflows.

Across the contribution chapters, the thesis combines quantitative analysis, qualitative assessment of review content, and tool construction. Quantitative analyses are used to compare distributions, study relationships among variables, and develop predictive models. Qualitative analyses are used to complement the quantitative perspective through review-quality grading and comment coding. Tool construction is used to operationalize the resulting methodology into reusable support for continuous MR data collection and analysis.

Each chapter operationalizes one or several of the specific objectives above by deriving metrics from MR traces, applying appropriate analytical methods, and interpreting the results from a SPI perspective. The final tooling contribution integrates these elements into a reusable workflow for researchers and industrial partners.

Contributions

This thesis makes the following contributions:

- C1. A method for leveraging MR traces as an observability instrument for DevOps process behavior. This contribution provides the empirical foundation of the thesis by showing that MR traces capture not only peer review activity, but also broader signals of environmental disruption, technological transition, organizational prioritization, and review-interaction dynamics that shape workflow behavior. [O1, O3]
- C2. A metric for measuring peer review effort based on the amount of post-submission changes, independent of review time. Grounded in the perspective established in C1, this contribution shows that post-submission changes capture a distinct dimension of review effort, affect a substantial proportion of MRs, and can be predicted early with strong performance, reaching an AUC of up to 0.88. [O1]

- C3. An analysis of the variation of peer review effort and quality across merge request types (development, configuration, and documentation), including the amplifying effect of bug presence. Building on C2, this contribution finds that configuration-related MRs generally receive less peer review effort than development MRs, while exhibiting distinct engagement and self-management patterns as well as stronger bug-related increases in review intensity and quality. [O1, O2]
- C4. A taxonomy and automated detection method for MR workflow deviations, enabling deviation-aware peer review effort analysis. This contribution shows that deviations are frequent, affecting up to 37% of MRs, and can be detected with up to 91% accuracy using few-shot learning. It also shows that deviations alter both the interpretation of the effort measures introduced in C2 and the ranking of influential predictive factors, with impacts that vary across the MR categories defined in C3. [O2]
- C5. An analysis linking upstream task estimation conditions to downstream peer review effort in industrial DevOps workflows. Extending the core effort-analysis perspective established in C2–C4, this contribution shows that a non-trivial share of downstream review burden can be traced back to upstream estimation inaccuracies, particularly underestimation in small tasks. [O3]
- C6. A predictive model for recommending safe self-approval of MRs at creation time, based on their relationship to CI/CD delivery reliability. As a downstream complement to C5, this contribution shows that self-approval is not uniformly associated with higher delivery risk and that creation-time recommendation models can identify MRs that are likely to be safely self-approvable with an AUC above 0.82. [O2, O3]
- C7. A mechanism for continuous and reproducible DevOps-pipeline data collection and analysis across Git-based platforms, implemented as the RevMine tool. Integrating the methodological and practical needs identified across C1–C6, this contribution operationalizes the thesis workflow into a reproducible tool that covers the three pipeline stages studied in the

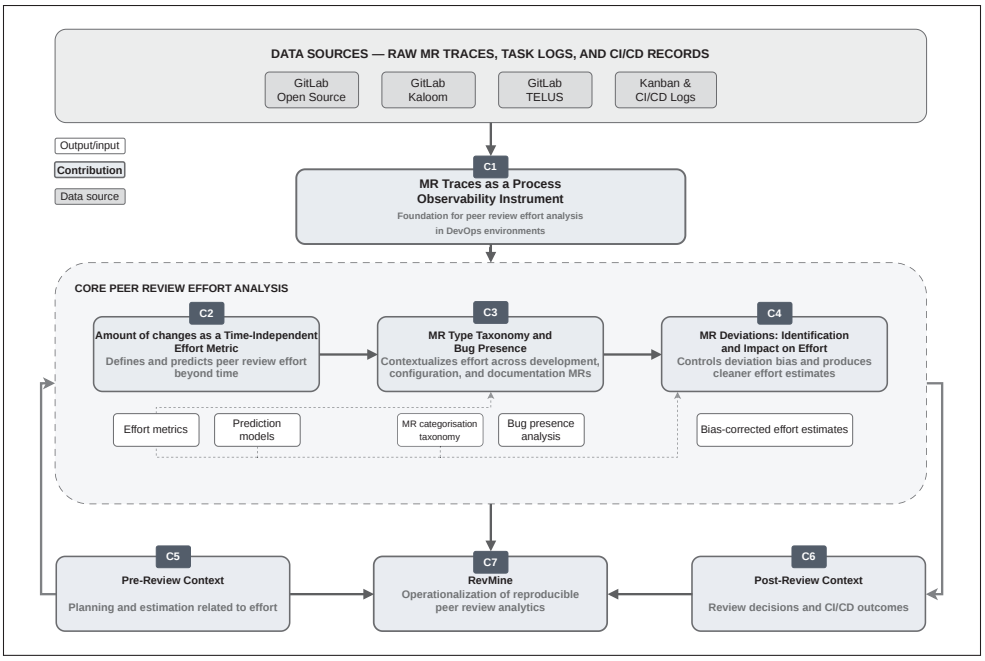


Figure 1 Overview of the thesis framework and its main contributions (C1–C7)

thesis — Kanban-style planning, peer review, and CI/CD delivery — and exposes cross-stage joins (issue → MR → pipeline) as first-class analytical primitives, supporting both empirical research and industrial transfer. [O4]

Figure 1 summarizes the overall framework of the thesis and illustrates how these contributions collectively support the analysis and optimization of peer review effort. Table 1 complements this overview by summarizing the associated publications and their corresponding replication packages for each contribution.

Thesis outline

The remainder of this thesis is organized as follows. Chapter 1 presents the industrial context and collaboration. Chapter 2 reviews the relevant background and related work. Chapter 3 describes the shared data collection infrastructure used across the empirical chapters. Chapter 4 establishes peer review traces as an observable process signal in DevOps settings and provides

Table 1 Traceability of thesis contributions to research papers and replication packages

C	Associated Paper Title	Venue / Status		
C1	Leveraging Merge Request Data to Analyze DevOps Practices: Insights from a Networking Software Solution Company	Published in the 14th International Conference on Software Engineering and Applications (SEA 2025)		-
C2	Understanding the Amount of Changes Required for Merge Request Acceptance: An Empirical Study	Published in the 16th IEEE International Conference on Software Engineering and Service Science (ICSESS 2025)		
C3	Do SDN Configuration Changes Get Reviewed Differently? An Empirical Study at TELUS	Published in the Empirical Software Engineering journal (EMSE)		
C4	Are All Code Reviews the Same? Identifying and Assessing the Impact of Merge Request Deviations	Published in the 41st IEEE International Conference on Software Maintenance and Evolution (ICSME 2025); extended to the Empirical Software Engineering journal (EMSE)		
C5	Do Size Estimates Predict Delivery? An Empirical Analysis of Task Estimation in Industrial DevOps	Accepted in the 42nd IEEE International Conference on Software Maintenance and Evolution (ICSME 2026)		
C6	Towards Risk-Aware Peer Review Effort Reduction in Industrial DevOps	Target the 49nd IEEE International Conference on Software Engineering (ICSE 2027), to be extended to a journal		
C7	RevMine: An LLM-Assisted Tool for Code Review Mining and Analysis Across Git Platforms	Published in the 35th IEEE International Conference on Collaborative Advances in Software and Computing (CASCON 2025); extended to the Empirical Software Engineering journal (EMSE)		

the empirical foundation for the thesis (C1). Chapter 5 defines and validates the amount of changes required to complete peer review as a time-independent measure of peer review effort and presents predictive models for identifying MRs likely to require substantial post-submission changes (C2). Chapter 6 examines how peer review effort, engagement, and review quality vary across development, configuration, and documentation MRs, including the effect of bug presence (C3). Chapter 7 characterizes MR deviations and analyzes their impact on effort measurement and cross-type comparisons (C4). Chapter 8 studies the relationship between upstream planning quality and peer review effort (C5). Chapter 9 connects peer review decisions to downstream CI/CD reliability (C6). Chapter 10 presents RevMine as a tooling contribution that operationalizes the analytical pipeline proposed in this thesis and extends it across the full DevOps pipeline by integrating Kanban-style planning data and CI/CD delivery data alongside peer review data (C7). Finally, Chapter 10.10 provides a global discussion, consolidated threats to validity, and overall conclusions.

CHAPTER 1

INDUSTRIAL CONTEXT AND RESEARCH COLLABORATION

1.1 Introduction

This thesis was conducted in close collaboration with two industrial partners, Kaloom and TELUS, within a broader collaborative research project on Software-Defined Networking (SDN), DevOps, and process improvement. These partnerships were essential to the thesis for two main reasons. First, they provided access to realistic industrial DevOps environments in which peer review, planning, integration, and delivery activities are performed under operational constraints. Second, they enabled a continuous exchange between academic analysis and industrial practice, ensuring that the research questions, metrics, and interpretations remained grounded in real organizational needs.

Beyond data access, the collaboration also shaped the way the research was conducted. The empirical studies reported in this thesis were discussed regularly with industrial stakeholders through recurring meetings, intermediate presentations, and broader project follow-up sessions involving both academic and industrial participants. As a result, the thesis was developed not only as an academic investigation of peer review effort, but also as part of a larger effort to improve the understanding and analysis of SDN DevOps processes in practice. This chapter provides the reader with the context necessary to understand the industrial setting in which the thesis was conducted, the structure of the collaborative project to which it contributed, and the interaction model that sustained the partnership throughout the research.

1.2 Industrial Partners

1.2.1 Kaloom

Kaloom is a Canadian company operating in the field of Software-Defined Networking. Its main technological focus in the context of this project is the *Kaloom Software Defined Fabric* (KSDF),

Figure 1.1 Kaloom company logo

an infrastructure designed to enable data centre providers to deploy and operate data centres that are more powerful, more flexible, and less costly than traditional data centre architectures.

From the perspective of this thesis, Kaloom provides a particularly relevant setting for studying DevOps process traces because its development activities involve large-scale software and infrastructure engineering, frequent integration activity, and a substantial use of GitLab-based workflows. These characteristics make it an especially suitable context for studying MRs not only as review artifacts, but also as process traces that reflect coordination, branch usage, automation, and workflow variation.

Kaloom was particularly important for the early parts of this thesis concerned with MR process observability and workflow deviations. Its repositories offered a rich and diverse industrial environment in which non-standard MR workflows could be observed and analyzed in practice. The breadth of projects and technical groups involved also made it possible to examine MR traces across heterogeneous industrial development contexts.

1.2.2 TELUS

Figure 1.2 TELUS company logo

TELUS is a major Canadian telecommunications company whose collaboration in this research project centered on SDN-enabled service delivery and automation. The industrial context considered in this thesis involves a *Network-as-a-Service* solution based on *SD-WAN*, with the broader objective of maximizing the use of cloud computing.

A central environment in this collaboration is the *TINAA* ecosystem, short for *TELUS Intelligent Network Analytics and Automation*. TINAA is an SDN solution developed to support in-house engineering activities and to facilitate the realization and operation of TELUS's current and future services. These services include, among others, the transmission of internet traffic, mobile telephone calls, and cable television programs. In this context, software and configuration artifacts are directly tied to operational network behavior, which makes the TELUS environment especially relevant for studying peer review in configuration-intensive SDN settings.

For this thesis, TELUS was particularly important for the analyses related to artifact-sensitive reviewing — especially the comparison of configuration, development, and documentation MRs. Because TELUS operates in an SDN environment where configuration artifacts are central to service realization and operation, its repositories provided a strong industrial basis for examining how peer review effort and quality vary across MR types. The TELUS context was also important for the study of upstream planning conditions, since task histories and workflow transitions could be reconstructed from GitLab issue and Kanban traces.

1.3 Global Collaborative Research Context

1.3.1 Project Structure and Work Packages

The work reported in this thesis was carried out within a larger collaborative project structured around four institutions — École de Technologie Supérieure (ÉTS), McMaster University, and Queen's University on the academic side, alongside Kaloom and TELUS as the industrial partners. The project was organized around four work packages, each addressing a distinct dimension of SDN DevOps research:

1. **WP1 — Process Capture:** Develop a framework for capturing SDN DevOps processes.
2. **WP2 — Measurement and Analysis:** Develop support for measurement and analysis of SDN DevOps processes.
3. **WP3 — Infrastructure Support:** Provide DevOps support for SDN infrastructure.
4. **WP4 — Application Support:** Provide DevOps support for SDN applications.

This thesis is primarily situated within **Work Package 2 (WP2): Measurement and Analysis of SDN DevOps Processes**. The goal of WP2 was to develop techniques and tools to assess software processes using criteria related to a company's broader corporate objectives. In particular, WP2 focused on two central criteria: (i) the time required to deliver a new or updated service, and (ii) the number of defects found during use of the software by customers or end users.

To support these objectives, WP2 aimed to work closely with the industrial partners in order to determine which elements of the process have the strongest impact on these criteria and how they can be improved most effectively. The work package therefore emphasized the identification of actionable process elements, the definition of relevant metrics, the extraction of data from DevOps tools, the computation of process indicators, and the development of methods to analyze and communicate these indicators to different stakeholders.

1.3.2 WP2 Activities and Their Alignment with the Thesis

More specifically, WP2 was organized around the following activities:

1. **Definition of process metrics.**
2. **Extraction of data from DevOps tools.**
3. **Computation of process metrics.**
4. **Methods to support the analysis of process metrics.**
5. **Methods to communicate process metrics and make them actionable.**

The contributions of this thesis align directly with these activities. Table 1.1 summarizes how each thesis contribution maps onto the WP2 activities described above.

Table 1.1 Alignment between thesis contributions and WP2 activities (A.n)

Thesis contribution	**WP2 activity**
MR traces as an observability instrument for DevOps process behavior	A.2 — Extraction of data from DevOps tools
Amount of post-submission changes as a time-independent peer review effort metric	A.1 — Definition of process metrics A.3 — Computation of process metrics
Variation of peer review effort and quality across MR types, including the effect of bug presence	A.4 — Methods to support metric analysis
Taxonomy and automated detection of MR workflow deviations for deviation-aware effort analysis	A.1 — Definition of process metrics A.4 — Methods to support metric analysis
Linking upstream task estimation conditions to downstream peer review effort	A.4 — Methods to support metric analysis A.5 — Methods to communicate metrics
Predictive model for safe MR self-approval based on CI/CD delivery reliability	A.5 — Methods to communicate metrics
RevMine: LLM-assisted tool for reproducible DevOps-pipeline data collection and analysis (Kanban, peer review, CI/CD) across Git-based platforms	A.2 — Extraction of data from DevOps tools A.5 — Methods to communicate metrics

1.4 Collaboration Model with the Industrial Partners

The collaboration with Kaloom and TELUS was not limited to repository access or periodic validation of results. Instead, it followed a continuous and structured interaction model intended to maintain close alignment between the academic work and industrial priorities throughout the full duration of the research.

At the operational level, **bi-weekly meetings** were organized with the industrial partners to present research progress, discuss intermediate findings, validate assumptions, and refine the next steps of the work. These regular meetings played an important role throughout the thesis. They made it possible to continuously confront analytical results with industrial interpretation, to ensure that the selected metrics remained meaningful in practice, and to adapt the research focus when necessary based on partner feedback.

In addition to these recurring meetings, the broader collaborative project included **face-to-face meetings twice per year** with the full set of academic and industrial participants, including collaborators from McMaster University and Queen's University. These meetings provided a higher-level forum to present the overall advancement of the project, discuss the progress of

the different work packages, and coordinate research directions across institutions and partners. They also supported cross-fertilization between the different parts of the project by allowing each work package to situate its findings within the broader SDN DevOps research program.

This collaboration model was important for the thesis in several ways. First, it ensured that the research remained strongly grounded in industrial practice rather than developing independently of real operational concerns. Second, it facilitated access to domain knowledge needed to interpret process traces and repository behavior correctly — particularly for SDN-specific configuration artifacts and TINAA workflow conventions at TELUS. Third, it enabled iterative validation of metrics, assumptions, and findings rather than limiting industrial involvement to the beginning or end of the research cycle.

1.5 Role of the Thesis Within the Collaboration

Within the broader collaborative project, this thesis contributed mainly to the measurement and analysis dimension represented by WP2. Its role was to help define, extract, compute, and analyze process-related information derived from GitLab traces in order to better understand how SDN DevOps processes operate in practice and how they may be improved.

This role is directly reflected in the thesis's seven contributions. The work on MR traces as an observability foundation aligns with the objective of capturing relevant process signals from operational repositories. The work on peer review effort metrics aligns with the objective of defining and computing meaningful, time-independent indicators. The analyses of MR types and workflow deviations contribute to understanding which contextual factors influence the interpretation of these indicators. The connections to upstream planning and downstream CI/CD outcomes extend this effort toward actionable process analysis that can inform industrial decision-making.

In this sense, the thesis does not stand apart from the collaborative project but rather forms one of its analytical pillars. It contributes empirical methods and findings intended to help identify where process inefficiencies emerge, how they can be characterized more precisely, and

how measurement can support more informed process improvement decisions in SDN DevOps environments.

1.6 Conclusion

This chapter presented the industrial context in which the thesis was conducted. It introduced Kaloom and TELUS as the two industrial partners whose SDN-oriented DevOps environments provided the empirical basis for much of the work. It situated the thesis within the broader collaborative research project and more specifically within WP2 on the measurement and analysis of SDN DevOps processes. It also presented a mapping between the thesis's seven contributions and the five WP2 activities, showing how each contribution serves a concrete analytical role within the project.

Finally, the chapter described the collaboration model that sustained the research throughout its duration, including bi-weekly operational meetings with industrial partners and twice-yearly face-to-face meetings involving all academic and industrial participants. Together, these interactions ensured that the thesis remained closely connected to practical industrial concerns while contributing rigorous empirical analysis to the broader SDN DevOps research program.

CHAPTER 2

BACKGROUND & RELATED WORK

2.1 Introduction

This thesis studies peer review effort as a software process phenomenon in DevOps environments. It examines peer review not only as a local quality assurance activity, but also as a rich process trace through which broader development behavior can be observed. In this perspective, prior work is relevant along several complementary dimensions: the role of peer review mechanisms in modern development workflows, the factors that shape peer review dynamics and effort, the quality and usefulness of review feedback, the influence of artifact type on review behavior, the treatment of workflow deviations, the role of planning and estimation in software workflows, the relation between peer review and downstream delivery contexts, and the tool support available for mining and analyzing review data.

This chapter reviews that literature and positions the thesis accordingly. Rather than organizing the discussion by individual chapter contributions, we synthesize prior work into five broader themes that collectively motivate the thesis's research questions.

2.2 Peer Review Mechanisms, Dynamics, and Effort

2.2.1 Peer Review in Modern DevOps Workflows

Peer review mechanisms play a central role in modern software development by structuring how proposed changes are discussed, revised, and integrated. In Git-based workflows, this process is commonly supported through MRs or Pull Requests (PRs), while Gerrit offers a more centralized, commit-oriented review model. These mechanisms provide a shared space for submitting changes, requesting feedback, discussing design or implementation issues, and deciding whether a change should be integrated into the main codebase. They also serve as an entry point for automation, including static analysis, testing, and bot-assisted review support (Tufano, Pascarella,

Tufano, Poshyvanyk & Bavota, 2021). As a result, peer review traces record not only a quality gate, but also a rich sequence of interactions between contributors, branches, automation, and process decisions.

Several studies have treated peer review as an important unit of analysis in software engineering. At a general level, peer-reviewed changes are associated with higher software quality, improved readability, and lower defect proneness (McIntosh *et al.*, 2014; Bavota & Russo, 2015; Tufano *et al.*, 2021). In industrial contexts, peer review is also embedded in broader DevOps workflows, where submitted changes move through automated checks, human review, integration decisions, and deployment-related activities. This integration makes peer review traces especially useful for studying software process behavior rather than only review outcomes.

The present thesis builds on this view by treating MR traces as observable process data. In contrast to work that focuses primarily on verification at the level of the reviewed change, this thesis argues that MRs also capture broader workflow signals related to coordination, process transitions, automation practices, and delivery conditions.

2.2.2 Time to Complete Peer Review, Efficiency, and Predictive Models

A large body of research has sought to analyze and improve the efficiency of peer review. One of the most studied dimensions is time to complete peer review as a peer review effort proxy. Several works developed predictive models for estimating the time needed to complete peer review and identifying the factors associated with that metric (Chouchen *et al.*, 2023; Maddila *et al.*, 2023; Hasan *et al.*, 2023; Thongtanunam *et al.*, 2017b; Islam *et al.*, 2022). For example, Chouchen *et al.* (2023) proposed machine learning models to predict time to complete peer review and explain its influential factors, while Maddila *et al.* (2023) examined review delay and the actors that block progress. Hasan *et al.* (2023) showed that shorter time-to-universal-first-response is generally associated with faster PR completion, although bot-first responses appear less informative than human involvement. Bosu & Carver (2014) similarly found that established developers receive faster initial feedback, which contributes to shorter time to complete peer review.

Other studies examined social, technical, and structural factors associated with review dynamics. Jiang, Adams & German (2013) showed that the number of reviewers affects time to complete peer review, while Thongtanunam *et al.* (2017b) investigated patch characteristics that attract more reviewer participation. Baysal, Kononenko, Holmes & Godfrey (2016) found that both technical and non-technical factors, such as team relationships and organizational structure, affect time to complete peer review. Studies also explored related inefficiencies, such as duplicate contributions (Li *et al.*, 2020) and abandoned pull requests (Khatoonabadi, Costa, Abdalkareem & Shihab, 2023), both of which increase wasted effort. Predictive work has further addressed whether submitted changes are likely to be merged or abandoned, with the goal of reducing delays or wasted review activity (Fan *et al.*, 2018; Islam *et al.*, 2022). Finally, Golzadeh, Decan, Legay & Mens (2021) examined bot-generated comments and highlighted both their utility for accelerating reviews and their limits for nuanced decision-making.

This literature establishes time to complete peer review as a useful and widely adopted proxy for effort. However, it also exposes an important limitation: temporal measures conflate actual review-related work with waiting time, prioritization, availability constraints, and organizational circumstances. This thesis builds directly on this limitation by arguing that peer review effort should not be approximated through time alone. Chapter 5 specifically extends this literature by introducing the amount of post-submission change as a time-independent and change-oriented indicator of peer review effort, thereby complementing the dominant temporal view found in prior work.

2.2.3 Review Feedback Quality and Usefulness

Beyond duration and participation, prior work has also examined the quality and usefulness of review feedback. This line of work recognizes that review value depends not only on whether feedback is provided, but also on whether it is understandable, actionable, and relevant to the change under review. Hasan, Iqbal, Islam, Rahman & Bosu (2021) proposed an automated approach for assessing peer review quality, while Doğan & Tüzün (2022) studied review smells and found that at least one smell was present in a large proportion of the reviews they analyzed.

Yang, Xu, Zhang, Zhang & Bacchelli (2023) introduced *EvaCRC*, an explainable framework for evaluating review comment quality with human-understandable justifications. Turzo & Bosu (2024) showed that the perceived usefulness of review comments depends not only on technical content, such as defect identification and improvement suggestions, but also on linguistic clarity and politeness.

Predictive studies also investigated what distinguishes useful from non-useful comments. Rahman, Roy & Kula (2017) proposed *RevHelper*, a model using textual features and reviewer experience to classify review comments by usefulness. Review quality has likewise been examined indirectly through review outcomes. Charoenwet, Thongtanunam, Pham & Treude (2024) studied security-related comments in large review datasets and showed that such feedback may lead to fixes, acknowledgments, or remain unresolved. Together, these studies show that review quality cannot be reduced to comment count or comment presence alone.

This thesis builds on that motivation in Chapter 6, where peer review quality is examined not only through quantitative measures but also through qualitative analysis of review comments across artifact types. In doing so, it connects the literature on review dynamics with the literature on review feedback quality and shows that both are necessary for understanding peer review effort in context.

2.3 Artifact Sensitivity, Configuration Changes, and Industrial Contexts

2.3.1 Heterogeneity Across Artifact Types

A growing body of work shows that peer review is not uniform across all changes. Instead, reviewers allocate attention differently depending on the artifact being modified, the perceived risk of the change, and the nature of the work. Nejati *et al.* (2023) analyzed build specification reviews and found that such changes often receive less attention than production and test code, despite carrying substantial defect risk. Spadini *et al.* (2018) showed that production code tends to receive more thorough review than test code when both are modified. Thongtanunam *et al.*

(2015) found that defective files may receive less rigorous review, and that issues raised during review often target evolvability concerns such as clarity and documentation rather than direct defect fixing. These studies support the broader idea that review behavior is artifact-sensitive.

Research has also examined change intent as a source of heterogeneity. Wang, Bansal, Nagappan & Philip (2019); Wang, Bansal & Nagappan (2021) studied reviews with large review effort, operationalized through multiple iterations, and showed that certain change intents, such as refactoring, are more likely to require substantial review effort. Their results also showed that intent-aware predictive models can outperform models that do not consider intent explicitly. However, their notion of effort differs from the one adopted in this thesis: they study review iterations rather than the amount of post-submission modification, and they focus on intent categories rather than broader process context.

The present thesis extends this artifact-sensitive perspective in several ways. First, it argues that configuration changes deserve explicit treatment rather than being absorbed into a general population of reviewed changes. Second, it distinguishes between configuration-dominant and configuration-inclusive MRs, thereby recognizing that configuration-related work may appear either as the main substance of a change or as one component of a mixed artifact change. Third, it examines review quality and effort together rather than focusing only on timing or iteration count. These extensions are developed mainly in Chapter 6.

2.3.2 Configuration Artifacts, IaC, and SDN Contexts

Given that both industrial partners in this thesis operate in the networking domain, configuration artifacts receive particular attention throughout this work. The importance of configuration artifacts has received increasing recognition in software engineering. Prior research has shown that configuration errors are a major cause of real-world system failures (Yin *et al.*, 2011). Infrastructure-related artifacts have also been shown to exhibit complex faults and recurring security issues, reinforcing the view that they should be treated as first-class software entities rather than peripheral files (Drosos, Sotiropoulos, Alexopoulos, Mitropoulos & Su, 2024;

Rahman, Rahman, Parnin & Williams, 2021; Dalla Palma *et al.*, 2024). These findings are particularly relevant in environments where configuration directly shapes system behavior.

A related line of work has begun to study review practices for configuration-centered artifacts and Infrastructure-as-Code (IaC). Bessghaier, Ouni, Sayagh, Chouchen & Mkaouer (2025) reported that review practices for IaC changes differ from those commonly observed for traditional software artifacts and explicitly highlighted the scarcity of research in this area. This work is among the closest antecedents to the present thesis. However, the current thesis extends that line in several directions: it focuses specifically on SDN configuration changes rather than general IaC, distinguishes configuration-dominant from configuration-inclusive MRs, combines quantitative review dynamics with qualitative comment analysis, and further isolates bug-related versus non-bug-related configuration changes.

The SDN context further strengthens the need for dedicated study. SDN decouples network control logic from forwarding devices and centralizes control in software controllers (Kreutz *et al.*, 2014; Nunes, Mendonca, Nguyen, Obraczka & Turletti, 2014). In such systems, configuration files do not merely tune peripheral aspects of behavior; they directly encode routing policies, access rules, service chains, and other operationally critical behavior (Benzekki, El Fergougui & Elbelrhiti Elalaoui, 2016). As a result, small configuration changes can have system-wide consequences that are not proportional to diff size and that may be difficult to validate locally. Despite these characteristics, prior SDN research has focused mainly on architecture, protocols, and verification rather than on collaborative peer review processes for SDN configuration changes. This thesis addresses that gap by empirically studying configuration reviews in an industrial SDN environment and comparing them with development and documentation reviews.

2.3.3 Industrial Review Practices and Automation

Peer review has also been widely studied in industrial settings. Bosu, Greiler & Bird (2015) conducted a large-scale study at Microsoft and identified several factors associated with comment

usefulness, including reviewer tenure and review scope. Jureczko, Kajda & Górecki (2020) compared tool-assisted and over-the-shoulder review techniques and showed that tool-assisted reviews produce higher comment volumes. AlOmar, AlRubaye, Mkaouer, Ouni & Kessentini (2021) studied the review of refactoring changes at Xerox and reported difficulties in documenting change intent and ensuring correct reviewer understanding. In parallel, automated review support has become common in industrial workflows, with examples including review bots at Samsung (Kim *et al.*, 2022), automated analysis at Google (Sadowski, Van Gogh, Jaspan, Soderberg & Winter, 2015), and automated support at Microsoft (Bird, Carnahan & Greiler, 2015).

Generative AI and large language models have further expanded automation possibilities in peer review. Pornprasit & Tantithamthavorn (2024) compared fine-tuning and few-shot prompting approaches and showed that few-shot methods can perform well in low-data settings. Haider, Mostofa, Mosaddek, Iqbal & Ahmed (2024) showed that augmenting prompts with semantic code metadata can significantly improve generated review comments. Jaoua, Sghaier & Sahraoui (2025) proposed hybrid pipelines combining large language models with static analyzers, while Tufano *et al.* (2022) and Li *et al.* (2022a) applied T5-based models to automated review generation. These works demonstrate the promise of AI for review automation. However, they generally assume standard review workflows and do not account for cases in which the MR mechanism is used for atypical or non-standard purposes. This thesis complements that literature by focusing not on automated comment generation, but on the quality of the analytical dataset itself, which Chapter 7 addresses through the study of workflow deviations.

2.4 Workflow Deviations, Dataset Quality, and Process Observability

2.4.1 Dataset Preprocessing and Analytical Validity

Many empirical studies on peer review rely on data preprocessing to improve analytical validity. Common practices include removing outliers, performing correlation and redundancy analysis, filtering inactive cases, and excluding instances that do not fit a given target variable

definition (Chouchen *et al.*, 2023; McIntosh, Kamei, Adams & Hassan, 2016a; Fan *et al.*, 2018; Islam *et al.*, 2022; Li *et al.*, 2020; Khatoonabadi *et al.*, 2023). Other studies focused more directly on cleaning review-related datasets. Liu, Lin & Thongtanunam (2025) used large language models to clean review comments, Liang *et al.* (2023) proposed a semantic approach for cleaning comment-update datasets, Krutauz, Dey, Rigby & Mockus (2020) used clustering to address multicollinearity, and Gupta & Sundaresan (2018) filtered non-actionable comments using regular expressions. These efforts all aim to improve the quality of the data on which analysis or automation is built.

However, most of this work treats data quality in statistical or textual terms rather than in process terms. That is, prior studies clean data by removing noisy values, redundant predictors, outdated comments, or non-actionable text, but they generally assume that the reviewed instances themselves represent comparable and standard peer review cases. The possibility that some MRs correspond to different workflow purposes rather than simply to noisy observations has received far less attention. Even work on bot detection, such as Golzadeh *et al.* (2021), which provides ground-truth data for identifying bot-generated activity, does not address the broader question of whether an MR as a whole represents a standard review workflow. This gap is central to Chapter 7, which extends dataset-quality research from statistical cleaning to workflow-aware analytical validity.

2.4.2 Branch Structure, Workflow Signals, and Process Observability

Some prior work has used branch information and review workflow structure to better understand peer review practices. Lal & Pahwa (2017) analyzed peer reviews using branches as a source of review information, while AlOmar (2025) studied refactoring branches and their impact on time to complete peer review, feedback patterns, and integration efficiency. Mukadam, Bird & Rigby (2013) examined which changes are reviewed, who reviews them, how long review takes, and what kinds of discussions occur, also using workflow and branch-related information. These studies indicate that structural information about the review process can reveal important patterns beyond the reviewed artifact itself.

The thesis builds on this idea at a broader level. Rather than using MR traces solely to explain review delay or participation, it argues in Chapter 4 that MR traces can serve as a process observability foundation in DevOps settings. Under this view, review traces reveal signals related to organizational priorities, branch strategies, automation, environmental disruption, and workflow adaptation. This extends prior review-mining work by making process observability, rather than only review performance, a central analytical goal.

2.5 Upstream Planning, Estimation, and Downstream Delivery

2.5.1 Effort Estimation in Agile and DevOps Workflows

Peer review effort does not occur in isolation from the planning conditions that precede it. Upstream estimation practices shape how work is scoped, prioritized, and assigned before any change reaches the review stage, and poor estimation has been shown to carry consequences that propagate through execution workflows. Effort estimation remains a core activity in software engineering because it influences planning, scheduling, and resource allocation (Ruk, Mahmood, Azmi, Firdaus Mohd Azmi & Gul, 2025b). In Agile workflows, estimation is commonly performed at the level of user stories or tasks using approaches such as T-shirt sizing, User Story Points, Planning Poker, and the Bucket System (Mallidi & Sharma, 2021b; Poženel, Fürst, Vavpotič & Hovelja, 2023). Parametric and data-driven approaches have also been adapted to Agile contexts (Chakravorty, Reddy & khan, 2025; Alsaadi & Saeedi, 2022).

Several studies also investigated automated task estimation. Catak, Durdu & Omurca (2024) proposed an NLP-based estimator for Agile requirements, William, Kumar, Chhabra & Vengatesan (2021a) linked estimation with task allocation in distributed Agile teams, and Thukkaram (2025) proposed MR-level effort estimation using repository and contributor features. Large language models have also been applied to effort estimation through zero-shot and optimized architectures (Radliński & Swacha, 2025; Miranda *et al.*, 2024; Cheemaa *et al.*, 2025), while Maiga, Bilgaiyan & Sagnika (2025) showed that smaller BERT-based models may outperform larger variants due to better generalization. Despite these advances, most work focuses on

predicting estimates rather than on understanding how estimates behave once work enters execution workflows.

The literature also highlights recurring limitations of estimation practices, including subjectivity, anchoring effects, inconsistent team calibration, and the misuse of estimates as rigid commitments (Mallidi & Sharma, 2021a; Sahni; Poženel *et al.*, 2023; Sandeep, Sánchez-Gordón, Colomo-Palacios & Kristiansen, 2022; Mateen & Malik, 2023). More broadly, software tasks are inherently uncertain (Albrecht & Gaffney, 1983), and temporal estimates may create pressure that prioritizes speed over quality (Goswami & Urminsky, 2020; Kim *et al.*, 2021; Arifin, Daengdej & Khanh, 2017). Related work on Kanban workflows also informs this perspective. Ahmad, Markkula & Oivo (2013) showed that empirical Kanban studies emphasize flow metrics such as lead time, cycle time, and throughput, while giving less attention to estimation accuracy or underestimation. Weflen, MacKenzie & Rivero (2022) proposed a probabilistic model for lead time estimation in Agile Kanban projects, again emphasizing forecasting rather than the behavioral consequences of estimates during execution.

This thesis extends such work in Chapter 8 by examining how task estimates and planning conditions relate to observed peer review effort and workflow outcomes, connecting upstream estimation behavior to downstream review activity rather than treating them as independent concerns.

2.5.2 Review Decisions and Downstream Delivery Reliability

The consequences of peer review decisions extend beyond the review stage itself. Once a change is approved and integrated, its quality and the rigor of the review process that preceded it have downstream implications for delivery pipelines, build stability, and system reliability. Several studies have examined the relationship between review practices and software quality outcomes. McIntosh *et al.* (2014) showed that inadequate review coverage is associated with higher post-release defect density, while Bavota & Russo (2015) found that review-related

metrics are among the strongest predictors of code quality. Bosu *et al.* (2015) observed that more experienced reviewers tend to produce feedback that leads to fewer downstream defects.

These findings motivate the perspective adopted in Chapter 9, which connects specific review decisions — including self-approval patterns and review scope — to CI/CD pipeline reliability. Rather than treating review quality as an outcome in itself, this chapter treats it as a predictor of downstream delivery conditions, thereby closing the loop from upstream planning through peer review to delivery.

2.6 Tool Support for Peer Review Mining and Analysis

2.6.1 Existing Repository Mining Tools

Several tools have been developed to support software repository mining and peer review analysis, each addressing a subset of the capabilities required for end-to-end empirical research. **PyDriller** (Spadini *et al.*, 2018) is a widely used Python framework for mining Git repositories at the commit level. It provides access to commit metadata, file diffs, and author information, but operates entirely at the local Git level and does not interface with the GitHub or GitLab REST APIs — meaning it cannot collect PR/MR metadata, reviewer assignments, discussion threads, or inline review comments. **GH Archive** provides a historical log of public GitHub event streams but is read-only, covers GitHub exclusively, and requires substantial post-processing to extract review-specific data. **GrimoireLab** is a comprehensive DevOps analytics platform that aggregates data from multiple sources including GitHub, GitLab, Jira, and Slack, but its configuration complexity, focus on aggregate dashboard metrics, and lack of researcher-oriented filtering make it poorly suited for fine-grained, reproducible empirical studies. **Boa** (Dyer, Nguyen, Rajan & Nguyen, 2013) is a domain-specific language and infrastructure for large-scale mining of open-source repositories, but it is limited to historical snapshots and does not support real-time or interactive data collection. **Perceval**, part of the GrimoireLab ecosystem, provides lower-level data fetching from multiple backends but lacks a unified analysis layer and requires significant scripting to produce research-ready datasets.

Crucially, none of these tools provides a natural language interface, an LLM-assisted interaction layer, or automatic metric dependency inference. As a result, researchers using any of these tools must still invest substantial effort in scripting, configuration, and data transformation before analysis can begin. Table 2.1 summarizes this comparison across five key capabilities that a peer review mining tool designed for empirical research should provide.

Table 2.1 Comparison of software repository mining tools across key capabilities relevant to empirical peer review research. ✓ = fully supported; ✗ = not supported; ~ = partial support

Tool	**Multi-Platform**	**Analysis Engine**	**NL Interface**	**LLM-Assisted**	**No-Code**
PyDriller (Spadini *et al.*, 2018)	✓	✗	✗	✗	✗
GH Archive	✓	✗	✗	✗	✗
GrimoireLab	✓	~	✗	✗	✗
Boa (Dyer *et al.*, 2013)	✗	~	✗	✗	✗
Perceval	✓	✗	✗	✗	✗
RevMine (ours)	✓	✓	✓	✓	✓

2.6.2 RevMine as a Complementary Layer

RevMine addresses all five dimensions simultaneously, positioning itself as a complementary layer that unifies data collection, transformation, visual exploration, and LLM-assisted interaction into a single reproducible pipeline. Unlike existing tools that require researchers to write custom extraction scripts for each study, RevMine allows users to express their intent in natural language and receive a validated, executable data collection plan in return — reducing both the technical barrier and the risk of inconsistent implementations across studies. Chapter 10 presents RevMine in full and evaluates it against the capabilities summarized in Table 2.1.

2.7 Conclusion & Positioning of the Thesis

Taken together, prior work provides a strong basis for studying peer review dynamics, review quality, artifact-sensitive review behavior, configuration-centered engineering, industrial review practice, dataset cleaning, and effort estimation. However, the existing literature remains fragmented in three important ways.

First, peer review is still predominantly studied as an isolated activity centered on timing, participation, or comment usefulness, without fully exploiting MR traces as broader process signals, and effort is still approximated mainly through temporal measures even though time may reflect waiting and organizational constraints rather than review-generated work alone. Second, while artifact sensitivity is acknowledged, configuration changes remain underexplored — especially in SDN environments — and data-cleaning research rarely addresses the possibility that some reviewed instances correspond to different workflow purposes rather than simply to noisy observations. Third, planning and estimation research has seldom been connected to peer review effort, the downstream delivery implications of review decisions remain underexplored within the same analytical framework, and tool support for reproducible end-to-end peer review analysis remains limited.

This thesis addresses these gaps through seven connected contributions. It establishes MR traces as a process observability foundation, introduces the amount of post-submission change as a time-independent indicator of peer review effort, examines how effort and review quality vary across MR types, studies workflow deviations and their analytical impact, relates upstream planning to peer review effort, connects review decisions to downstream delivery reliability, and provides RevMine as tooling support for reproducible peer review analytics. In this way, the thesis contributes not only to understanding peer review itself, but also to understanding how peer review effort should be interpreted within the broader software process.

CHAPTER 3

DATA COLLECTION

3.1 Introduction

This thesis relies on repository mining to study peer review effort and its surrounding software process context in DevOps environments. Because the thesis addresses several complementary questions—including peer review observability, effort measurement, artifact-sensitive review behavior, workflow deviations, upstream planning, and downstream process conditions—it draws on multiple datasets collected from both industrial and open-source GitLab repositories. Rather than treating these datasets as isolated chapter-specific resources, this chapter presents them as parts of a common data collection strategy centered on GitLab development traces.

Across the thesis, the main source of empirical evidence is the history of GitLab artifacts and events associated with MRs and issues. Since our industrial partners use GitLab and the MR mechanism as part of their peer review workflow, these traces provide a rich basis for analysis. They include review-related information such as MR metadata, commits, discussions, notes, reviewers, file changes, and diffs, as well as planning-related information such as issues, labels, assignments, workflow states, and issue histories. Together, these records offer a longitudinal and fine-grained view of how work is proposed, reviewed, revised, and advanced in both industrial and open-source DevOps settings.

Although each empirical chapter emphasizes the dataset that is most directly aligned with its objective, the broader data collection infrastructure is shared across the thesis. For example, the effort-oriented analyses are primarily reported on open-source projects because the proposed effort measure is first established in that setting, the artifact-type analyses are emphasized on the TELUS dataset because configuration-related reviewing is especially central there, and the deviation analyses are emphasized on the Kaloom dataset because workflow deviations were first prominently observed in that context. Nevertheless, the studies are not strictly isolated by dataset, and several chapters draw on additional datasets for replication, contrast, or validation.

3.2 Data Sources and Collection Infrastructure

To support the different empirical studies of the thesis, we rely on custom extraction tooling built on top of the GitLab API. A first version of this infrastructure was initially developed by Legault (2022), and later extended into a custom MR data extraction tool tailored to the needs of this thesis. The tool uses GitLab API endpoints to collect MR-related information, including metadata, commits, discussions, notes, and file modifications.[1] [2] [3] [4] [5] In studies that require linking reviewed changes to planning artifacts, the same infrastructure is also used to extract issue-level data and to establish associations between MRs and the issues they address.[6] To support transparency and reuse, the MR extraction tool is publicly available.[7]

The data collection strategy is intentionally broad. It supports the extraction of:

- MR metadata, such as identifiers, authors, creation dates, closure dates, and merge dates;
- commit-level information, including commit messages, code changes, and diffs;
- peer review artifacts, such as comments, discussions, notes, reviewers, and approvals;
- file-level change information, including additions, deletions, and touched files;
- issue-level information, such as titles, descriptions, labels, assignments, and state transitions;
- issue and workflow histories, enabling the reconstruction of task lifecycles in planning and execution environments.

These extracted traces form the empirical basis of the thesis and enable the study of peer review as both a local review activity and a broader software process phenomenon. For each repository, we retain the raw extracted data and derive case-specific metrics afterward, depending on the

[1] https://docs.gitlab.com/ee/api/merge_requests.html

[2] https://docs.gitlab.com/api/merge_requests/

[3] https://docs.gitlab.com/api/commits/

[4] https://docs.gitlab.com/api/discussions/

[5] https://docs.gitlab.com/api/notes/

[6] https://docs.gitlab.com/ee/api/issues.html

[7] https://gitlab.com/ets-devops/merge-requests/mr-analysis-tool

objectives of the chapter. This strategy avoids repeated extraction and recomputation, which are both time- and resource-consuming, while ensuring consistency across studies.

3.3 Industrial MR Datasets

A large part of the thesis is grounded in industrial GitLab repositories obtained through collaborations with Kaloom and TELUS. These industrial datasets provide access to realistic DevOps environments in which MRs are regularly created, reviewed, revised, and integrated under operational constraints.

3.3.1 Kaloom Dataset

The first industrial dataset supports the thesis's contributions on MR traces as a process observability foundation and on workflow deviations. Using the custom GitLab extraction tool, we collected MR data from multiple projects belonging to five major technical groups of our industrial partner Kaloom. The extracted data includes MR identifiers, creation and closure dates, commits, discussions, notes, reviewers, and file modifications. This dataset was designed to support the analysis of broader workflow signals observable through MR traces, including the effects of process changes, branch usage, review interaction patterns, and non-standard review workflows presented as deviations.

The studied projects are grouped into five major technical areas: Management Plane (MP), Control Plane (CP), Data Plane (DP), FPGA, and Platform (PF). Together, these groups aggregate 116 underlying sub-projects (20 in MP, 31 in CP, 16 in DP, 19 in FPGA, and 30 in PF) and cover a broad range of software and infrastructure responsibilities, including management of communication with the fabric, orchestration of packet and frame paths, forwarding behavior, reprogrammable hardware, and low-level platform management. This diversity is valuable for studying MRs as process traces across heterogeneous industrial development contexts.

Table 3.1 summarizes the main repository and change-profile characteristics of the Kaloom dataset using the same set of common metrics later used for the TELUS and open-source datasets.

Table 3.1 Repository characteristics and change profile of the studied Kaloom repositories

Repo	#MRs	#Com	Avg. ppl/MR	Add.	Del.	Churn	#Files
CP	8,396	17.3k	1.05	895k	450k	1.35M	7,733
DP	7,416	18.2k	1.07	1.11M	471k	1.58M	5,341
FPGA	735	4.09k	1.10	298k	150k	448k	4,876
MP	6,344	19.5k	1.04	1.21M	522k	1.73M	6,078
PF	4,004	6.99k	1.03	229k	109k	338k	1,895
Total	**26,895**	**66.0k**	**1.06**	**3.74M**	**1.70M**	**5.44M**	**25,923**

#Com: total number of commits.
Avg. ppl/MR: average number of contributors per MR.
Churn: additions + deletions aggregated over all MRs.

3.3.2 TELUS Dataset

Our second industrial dataset comes from TELUS and is used primarily for the thesis's analyses of artifact-sensitive peer review behavior, especially in a configuration-intensive SDN environment. For these studies, we collected data from five projects belonging to the TINAA GitLab group, an internal SDN software ecosystem maintained by TELUS. The selected projects were chosen in collaboration with engineering leads to ensure that they represent realistic and configuration-intensive development scenarios. They are central to the SDN platform's control and automation layers and follow modern DevOps practices, including regular peer review, automated testing, and continuous integration.

This dataset contains 8,495 MRs. For each MR, we extracted core metadata, commit-level information, review artifacts, and associated issue information. More specifically, the dataset includes MR identifiers, authors, creation and merge dates, commit messages, changed files, diffs, inline comments, discussions, reviewers, issue titles, issue descriptions, and labels. These traces provide the empirical basis for studying how peer review effort and quality vary across configuration, development, and documentation MRs, as well as for analyzing bug-related versus non-bug-related configuration changes.

An important methodological advantage of this dataset is that the selected projects share the same organizational setting, time window, GitLab-based workflow, and engineering policies.

Table 3.2 Repository characteristics and change profile of the studied TELUS repositories

Repo	#MRs	#Com	Avg. ppl/MR	Add.	Del.	Churn	#Files
NetApps	5,811	37.5k	2.78	3.51M	1.82M	5.33M	5,446
Platform	1,115	9.04k	2.11	846k	440k	1.29M	2,749
Infrastructure	363	1.11k	2.05	32.1k	17.7k	49.8k	787
CDAF	1,014	7.25k	2.25	1.01M	454k	1.46M	2,304
CNE-IP	192	1.01k	1.38	168k	83.2k	251k	647
Total	**8,495**	**55.9k**	**2.54**	**4.25M**	**2.81M**	**8.38M**	**11,933**

#Com: total number of commits.
Avg. ppl/MR: average number of contributors per MR.
Churn: additions + deletions aggregated over all MRs.

This consistency supports fair cross-project comparisons by reducing variability that could otherwise arise from differences in tooling, governance, or review procedure.

Table 3.2 summarizes the main repository and change-profile characteristics of the TELUS dataset using the same common metrics as in the Kaloom and open-source summaries.

3.4 Open-Source MR Datasets

To complement the industrial setting and improve the breadth of empirical observations, the thesis also uses open-source GitLab repositories. These repositories are particularly important for validating findings related to peer review effort beyond the industrial context and for studying MRs in projects with large review histories.

For the effort-oriented study of Chapter 5 and generalisability investigation, we collected MR data from four open-source GitLab projects with more than 3,000 MRs: Inkscape, Omnibus-GitLab, GitLab Runner, and Gitaly. These projects were selected because they provide sufficiently large MR histories and represent mature GitLab-based development workflows. Using the GitLab API, we extracted data related to MRs, commits, code changes, discussions, and review comments. The collected dataset is also available through the replication package.[8]

[8] https://figshare.com/s/632a15054ffb881ae72d

Table 3.3 Repository characteristics and change profile of the studied open-source repositories

Repo	#MRs	#Com	Avg. ppl/MR	Add.	Del.	Churn	#Files
GT	5,138	2,800	1.00	165k	92.8k	258k	8,164
OB	4,688	3,282	1.00	89.4k	62.8k	152k	6,761
GR	2,499	1,482	1.00	47.9k	41.7k	89.6k	3,106
Ink	3,643	1,689	1.00	102k	70.0k	172k	5,131
Total	**15,968**	**9,253**	**1.00**	**405k**	**267k**	**672k**	**23,162**

#Com: total number of commits.
Avg. ppl/MR: average number of contributors per MR.
Churn: additions + deletions aggregated over all MRs.
GT = Gitaly, OB = Omnibus-GitLab, GR = GitLab Runner, Ink = Inkscape.

These open-source projects play two main roles in the thesis. First, they support the study of amount of post-submission change as a time-independent indicator of peer review effort by providing diverse, MR-rich repositories outside the industrial setting. Second, they provide a basis for validating the broader applicability of effort-related findings that are initially observed in industrial contexts. In later analyses, these repositories are also reused as a validation context alongside the industrial datasets when appropriate.

Table 3.3 summarizes the main repository and change-profile characteristics of the open-source dataset using the same common metrics as in the industrial summaries.

3.5 Issue and Workflow History Data for Upstream Planning Analysis

The thesis also examines how upstream planning conditions relate to peer review effort. For this purpose, we collected issue-centered workflow data from TELUS GitLab repositories in order to reconstruct task lifecycles in a Kanban-based DevOps environment. In addition to issue snapshots, we extracted users and complete issue histories, including assignment changes, state transitions, size updates, and movements across Kanban columns.

This event-history perspective is important because current issue snapshots do not preserve the sequence of workflow changes through which a task progresses. At TELUS, workflow stages are represented using status labels such as `Status::To Do`, `Status::In Progress`, and

`Status::In Review`. By mining these historical changes, we can reconstruct task timelines and compute variables such as lead time, workflow exposure, execution duration, and waiting duration. After preprocessing, the final issue dataset contains 13,419 issues from 137 industrial projects, providing a large-scale and fine-grained view of task execution in practice.

This issue-history dataset supports the thesis's contribution on upstream planning and effort. More specifically, it enables the study of how task-level planning and estimation conditions relate to the effort later observed during peer review.

3.6 Role of the Collected Datasets in the Thesis

Although the thesis relies on multiple datasets, they are all collected through a unified methodological perspective: the use of GitLab-based process traces to study how work is planned, reviewed, revised, and advanced in DevOps environments. The different datasets therefore play complementary roles.

The Kaloom dataset primarily supports the study of MR traces as a process observability foundation and the analysis of workflow deviations. The TELUS dataset primarily supports the study of artifact-sensitive peer review behavior, especially in configuration-intensive SDN projects. The open-source MR datasets primarily support effort-oriented analyses and help assess the broader relevance of selected findings outside the industrial setting. Finally, the issue and workflow-history dataset supports the study of upstream planning conditions and task execution dynamics.

At the same time, these roles are not exclusive. Several findings are contrasted, replicated, or interpreted across more than one dataset. The main analyses reported in a chapter therefore reflect the dataset that best motivates the corresponding contribution, but the broader empirical strategy remains multi-dataset throughout the thesis.

Together, these datasets allow the thesis to connect peer review effort to its surrounding process context rather than analyzing peer review in isolation. This multi-dataset strategy is essential

to the thesis's central objective, namely, to understand peer review effort as a software process phenomenon in DevOps environments.

3.7 Conclusion

This chapter presented the data collection foundations of the thesis. Using custom extraction tooling built on the GitLab API, we collected MR-, issue-, and workflow-related traces from both industrial and open-source repositories. These datasets provide detailed information about changes, discussions, revisions, review activity, and task progression, thereby enabling the thesis's empirical analyses across several complementary contributions. In the following chapters, these datasets are used to examine MR traces as process signals, to characterize peer review effort beyond time-based measures, to study differences across MR types and workflow deviations, and to relate peer review effort to planning and execution conditions.

CHAPTER 4

LEVERAGING MERGE REQUEST DATA TO ANALYZE DEVOPS PRACTICES: INSIGHTS FROM A NETWORKING SOFTWARE SOLUTION COMPANY

Samah Kansab[1] , Matthieu Hannania[1] , Francis Bordeleau[1] , Ali Tizghadam[2]

[1] Department of Software Engineering and IT, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[2] TELUS, 25 York St, Toronto, Ontario, Canada M5J 2V5

Paper Published in the 14th International Conference on Software Engineering and Applications on May 2025

Preamble

This chapter studies whether MR traces can reveal more than review activity alone. Using 26,895 MRs from 116 sub-projects grouped into five technical groups at an industrial networking software company, we analyze four dimensions: environmental disruption (the COVID-19 shift to remote work), internal technological transition (the OpenShift migration), organizational priorities (branch management), and review interaction dynamics (bot versus human first comments). The findings show that MR traces make workplace change, process transitions, release prioritization, and automation-human interaction empirically visible, establishing MR data as an observability foundation for the effort-oriented analyses developed later in this thesis. This chapter is adapted from the paper published at SEA 2025. The thesis version revises the introduction, conclusion, and overall framing to align the chapter with the objectives of this thesis. Figures are kept identical in content to those of the original paper, but their graphical quality has been improved for the thesis version.

4.1 Introduction

Peer review data provides a valuable empirical basis for studying software processes in DevOps environments. In Git-based platforms, each MR produces a structured and timestamped record of a proposed change, including its submission, review interactions, revisions, automation activity, and final outcome. Because MRs are positioned between change preparation and integration,

their traces do not only describe peer review activity itself. They may also reflect broader process conditions, such as work prioritization, technological transitions, organizational constraints, and the interaction between human reviewers and automation.

This perspective is central to this thesis. As argued in the general introduction, peer review effort cannot be fully understood if review is treated as an isolated event. The work required during review may be shaped by upstream conditions, while review decisions may influence downstream integration and delivery behavior. Before studying peer review effort in detail, it is therefore necessary to establish whether MR traces can serve as a reliable observability instrument for analyzing software process behavior more broadly.

Prior research has used peer review data to study topics such as review latency, reviewer assignment, merge prediction, and comment usefulness (Hasan *et al.*, 2023; Chouchen *et al.*, 2023; Maddila *et al.*, 2023; Islam *et al.*, 2022). These studies have provided important insights into peer review as a technical and collaborative activity. However, less attention has been given to the broader process information embedded in MR traces, particularly their ability to reveal how teams respond to disruptions, adapt to internal process changes, encode organizational priorities, and coordinate human and automated review activity over time.

This chapter addresses this gap by examining whether MR traces capture process signals beyond peer review itself. We conduct a large-scale empirical study on Kaloom projects grouped into five technical groups at an industrial networking software company specializing in 5G and cloud-edge fabric technologies (see Section 3.3.1). The analysis considers four forms of process variation: an external disruption caused by the COVID-19 transition to remote work, an internal technological transition through the migration to OpenShift, organizational prioritization reflected in branch management practices, and local review interaction dynamics involving human and bot comments.

The results show that MR traces capture meaningful signals across these four dimensions. The COVID-19 transition is reflected in changes in working patterns, including increased activity outside regular hours. The OpenShift migration appears through temporary instability in review

and change-size indicators. Branch-level analysis shows that stable branches receive prioritized treatment, reflecting release-management concerns. Finally, first-comment dynamics reveal that bots accelerate initial responses, while human comments remain more strongly associated with review progression. Together, these findings establish MR traces as a foundation for the later chapters, where peer review effort is analyzed as part of a broader DevOps process rather than as an isolated review outcome.

The main contributions of this chapter are:

- an empirical demonstration that MR traces reveal the effects of external disruption on software work, using the COVID-19 transition to remote work as a natural experiment;
- evidence that internal technological transitions, such as a migration to OpenShift, are visible through review behavior and workflow-stability indicators over time;
- an analysis showing that branch management practices encode organizational release priorities, with stable branches receiving prioritized review treatment;
- a characterization of human-versus-automated first-comment dynamics in MR reviews, showing that bots accelerate initial response while human comments remain the main drivers of review progression.

4.2 Study Design

In this chapter, we aim to establish the broader observability role of peer review traces by examining whether data extracted from MRs can reveal meaningful signals beyond the peer review process itself. To achieve this objective, we design an empirical study structured around four research questions, each targeting a distinct dimension of DevOps process behavior as reflected in MR traces. Together, these questions allow us to assess whether peer review data can capture the effects of environmental disruption, internal technological transition, organizational prioritization, and local review interaction dynamics.

4.2.1 Research Questions

The chapter is guided by the following four research questions:

RQ1 **How have environmental changes, specifically the shift from office-based work to remote work during the COVID-19 pandemic, impacted workplace dynamics and productivity?** This question examines whether MR traces reflect changes in collaboration structure, work rhythm, and review effort across the transition to remote work. To address it, we analyze temporal variations in MR activity, collaboration-related indicators, and effort-related measures before, during, and after the disruption period.

RQ2 **Has the migration to OpenShift technology affected different aspects of the organization development process?** This question investigates whether a planned technological transition becomes visible in peer review traces. More specifically, we study how the OpenShift migration influenced MR volume, review efficiency, and workflow-related characteristics over the course of the transition.

RQ3 **How are different types of GitLab branches used and managed in an industrial DevOps environment?** This question focuses on branch management practices as a reflection of organizational priorities. We analyze how different branch categories are handled in the MR process, with particular attention to prioritization patterns, review speed, and the role of branch types in structuring development activity.

RQ4 **To what extent does the origin of the first comment—bot or human—influence the time to complete peer review?** This question examines the role of early interaction in review progression. More specifically, we assess whether the source of the first comment is associated with differences in time to complete peer review, and we study the relative influence of automated feedback, human intervention, and additional contextual factors on review efficiency.

4.2.2 Metric Extraction

This study relies on MR data collected from an industrial GitLab environment, as described in Section 3.3.1. After collecting the raw MR data, we extract a set of derived metrics used to characterize the different dimensions of the study. Some of these metrics are directly available from the GitLab platform, whereas others must be computed from raw event data. For example, the time to complete peer review is derived as the elapsed time between MR creation and MR closure, since this measure is not directly provided by the GitLab API. Because the four research questions address different process phenomena, they do not rely on exactly the same subsets of metrics. Instead, each question is analyzed using the metrics that are most relevant to its specific objective.

In total, we extract 24 metrics, summarized in Table 4.1. These metrics span three complementary dimensions:

- the *MR dimension*, which includes metrics describing the properties and temporal progression of individual MRs;
- the *project dimension*, which captures contextual information about the project in which the MR occurs;
- the *developer dimension*, which includes characteristics related to contributors involved in the review process.

These metrics provide a structured view of peer review traces and make it possible to analyze the four research questions from multiple process perspectives. As shown in Table 4.1, not all metrics are used in every research question; instead, each question relies on the subset of metrics most appropriate to its analytical objective.

4.2.3 Analysis Strategy

The analytical strategy varies according to the objective of each research question, but follows a common overall logic. First, the data is filtered and organized according to the temporal or contextual dimensions relevant to the question under study. In particular, for questions involving

Table 4.1 Metrics extracted from MR traces and used across the four research questions

Metric	Description	Justification for use
Week of creation ‡	The calendar week in which the MR was created.	Used to aggregate MR activity over time and compare weekly patterns across the COVID-19 disruption period.
Sprint of creation †	The development sprint in which the MR was created.	Used to analyze process evolution according to the organization's three-week development cycle during the OpenShift migration.
Week of closure ‡	The calendar week in which the MR was closed.	Used to examine weekly completion patterns and changes in closure activity over time.
Time to complete peer review ‡ † ∗ ⋆	The elapsed time between MR creation and MR closure.	Used as a core indicator of review progression and effort across all four research questions.
Hour of creation ‡	The hour of the day at which the MR was opened.	Used to study daily work rhythm and detect shifts in work timing.
Hour of closure ‡	The hour of the day at which the MR was closed.	Used to assess whether completion activity moved outside normal work schedules.
Presence of activity outside regular working hours ‡	A binary indicator showing whether the MR includes activity outside regular working hours.	Used to quantify the extent to which work expanded beyond standard office hours during remote work.
MR size † ∗	The total number of modified lines, computed as additions plus deletions in the MR.	Used to characterize the scale of a change and compare whether transition periods or branch types are associated with different MR sizes.
Number of added lines †	The number of lines added in the MR.	Used to distinguish the growth component of change during the migration analysis.
Number of deleted lines †	The number of lines removed in the MR.	Used to characterize the removal component of change and better interpret migration-related modifications.
Source branch ∗	The branch from which the MR originates.	Used to identify the development context of the change and analyze branch management practices.
Target branch ∗	The branch toward which the MR is directed.	Used to determine which branches receive prioritized review and handling.
Time to first comment ∗ ⋆	The elapsed time between MR creation and the first review comment.	Used to study early review responsiveness and the speed of initial feedback.
Number of discussion notes ∗	The number of discussion notes associated with the MR.	Used as an indicator of review interaction intensity and discussion volume.
Description length ⋆	The length of the textual description provided in the MR.	Used to capture how much contextual information is initially provided with the change request.
Whether it is the first MR in the project ⋆	A binary indicator showing whether the MR is the first one submitted in its project.	Used to control for atypical behavior associated with early project activity.
Presence of a hashtag in the title or description ⋆	A binary indicator showing whether the "#" symbol appears in the MR title or description.	Used to capture explicit references to issues, tickets, or work items.
Presence of a user mention in the title or description ⋆	A binary indicator showing whether the "@" symbol appears in the MR title or description.	Used to capture direct mentions of collaborators and explicit communication cues.
Number of commits at opening ⋆	The number of commits included in the MR at the time of opening.	Used to characterize the initial structural complexity of the proposed change.
Project age ⋆	The number of months between project creation and MR submission.	Used to account for project maturity when analyzing review behavior.
Number of open MRs in the project ⋆	The number of MRs already open in the same project at the time of analysis.	Used to capture the concurrent review workload within the project.
Number of prior merged MRs ⋆	The number of previously merged MRs in the project.	Used as an indicator of prior project activity and accumulated integration history.
Number of previous MRs ⋆	The number of all previously created MRs in the project.	Used to represent the historical review volume of the project.
Whether the author is also the reviewer ⋆	A binary indicator showing whether the MR author is also assigned as reviewer.	Used to capture a potentially atypical review configuration that may affect review progression.

‡ RQ1: Environmental disruption † RQ2: Internal process transition ∗ RQ3: Organizational priorities ⋆ RQ4: Review interaction dynamics

process evolution over time, we analyze MR data at the level of weeks or sprints, where a sprint corresponds to the company's three-week development cycle. This temporal structuring makes it possible to compare behavior across different periods and organizational conditions.

Second, we compute descriptive and comparative indicators such as means, medians, minima, and maxima in order to characterize trends and contrasts across conditions. Depending on the research question, these indicators are used to study aspects such as productivity, review effort, prioritization behavior, and interaction dynamics. For example, productivity may be reflected by the number of MRs created and closed over time, whereas effort-related behavior may be examined through measures such as time to complete peer review.

Finally, the specific analytical procedures used for each research question are detailed in the corresponding sections of the chapter. This organization allows the study design to remain coherent at the chapter level while preserving the methodological flexibility needed to address four distinct process dimensions.

4.3 Environmental Disruptions: COVID-19 Case Study

RQ1: How have environmental changes, specifically the shift from office-based work to remote work during the covid-19 pandemic, impacted workplace dynamics and productivity?

Approach

To investigate the impact of COVID-19 on various aspects of the DevOps process, we conducted a comparative analysis before and after the pandemic, focusing on the following key issues:

- **Productivity:** to have clearer vision, we conduct this analysis spanning the first 22 weeks of 2020. This period includes 11 weeks before and 11 weeks after the start of the pandemic-induced lockdown, which began at week 12. We assessed team productivity by examining the number of MRs created and ended each week before and after the COVID-19 outbreak, including transition week 12. Tracking the number of MRs over time helps us see how fast

development tasks are being merged and completed. More MRs created could mean that more work is being done, indicating higher productivity. But if there's a big decrease or change in the number of MRs completed, it could mean there are problems or delays in the development process. This analysis helps teams identify trends and issues in their DevOps workflow.

- **Efforts:** In assessing the impact of COVID-19 on developer effort, we study time to complete peer review on the 22 weeks, which represents the time it takes to complete a peer review. This involved analyzing the time taken to complete peer review both before and after the pandemic, as well as the number of successfully merged MRs. A longer time to complete peer review suggests that collaborators spent more time reviewing, providing feedback, and updating the code, indicating an increased effort to ensure code quality and accuracy. Alternatively, it could indicate delays in the peer review process, resulting in a slower overall timeline. Conversely, shorter time to complete peer review may indicate less effort (a simpler MR) in the review process, or they may reflect the prioritization of the work, necessitating a quicker completion. Therefore, as supported by previous research Chouchen *et al.* (2023); Maddila *et al.* (2023); Baysal *et al.* (2016); Jiang *et al.* (2013), we consider the time to complete peer review as a proxy to analyze the effort to complete the review process.
- **Temporal Autonomy:** With the transition to remote work, individuals have gained greater flexibility in their work schedules, allowing them to work during nights, weekends, or holidays. In this phase, we examine the impact of COVID-19 on developers' temporal autonomy by analyzing their activity on MRs before 8 a.m. and after 5 p.m. This includes actions such as creating or ending MRs, reviewing, and committing. For each week, we calculate the percentage of these activities occurring outside regular working hours compared to the total number of activities throughout the entire day. This analysis includes all data to study the persistence of the pandemic's impact on developers' habits years after the pandemic.

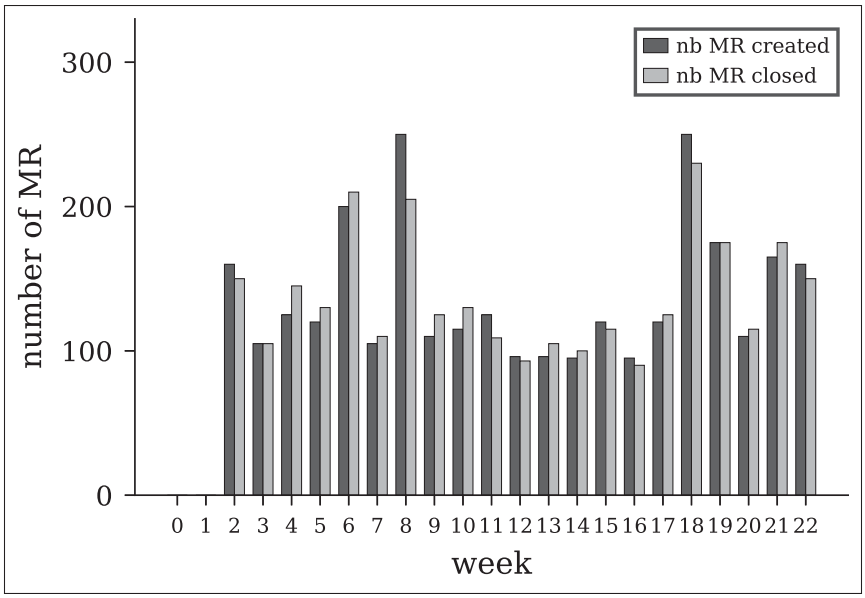


Figure 4.1 Number of MR created and ended per week from week 1 to 22

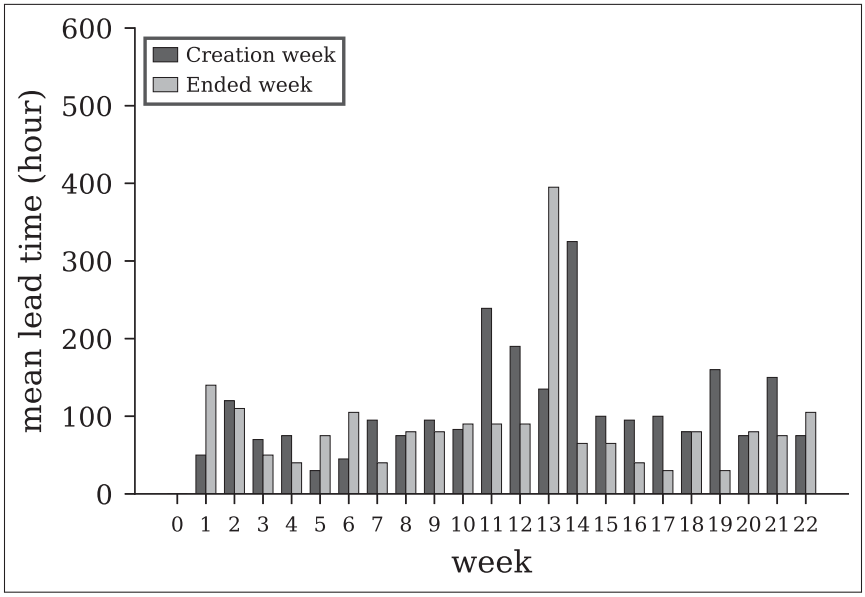


Figure 4.2 Mean time to complete peer review by week for MR opening and closing from week 1 to 22

Results

Despite the onset of the COVID-19 pandemic, the productivity of the teams of the studied company, as measured by the weekly number of MRs created and closed, remained relatively

stable. Figure 4.1 illustrates this trend, showing that there was no significant change in the volume of MRs despite the onset of the pandemic. Specifically, there were 125 MRs created in week 11 (the last week of work in the office), which decreased slightly to 96 in week 12, a decrease of 23.2%. Similarly, the number of MRs closed decreased slightly from 109 in week 11 to 93 in week 12, a decrease of 14.68%. Regarding week 13 (the first week after the restriction decision), we observe that the number of MRs created is the same as in week 12. We also notice that the number of MRs closed exceeds the number of MRs created in this week. However, we observe this pattern (number of closed MRs exceeding the number of created MRs) in other weeks prior to COVID-19, such as weeks 4, 9, and 10, indicating that this is not solely the impact of COVID-19. These results suggest that, contrary to expectations, the COVID-19 pandemic did not have a significant impact on the productivity levels of the teams when measure based on their MR activity.

Note: The first week of the year 2020 is empty. Because on this week, no developers have created (open) or end (close) any MR while it was January holidays.

Contrary to our initial findings, we observe a significant increase in time to complete peer review with the onset of COVID-19. This indicates either greater effort or more delays during the review process by team members during this period.. As shown in Figure 4.2, the mean time to complete peer review for MRs created in week 10 jump from about 3.47 days to about 9.95 and 7.92 days in weeks 11 and 12, respectively, coinciding with the transition to remote work. This trend is similar to that observed in week 2 (after the January holidays), confirming the impact of COVID-19 on the time to complete peer review of MRs created in weeks 11 and 12. Regarding the average time to complete peer review of closed MRs, those closed in week 13 have an average of about 16.44 days, which is the greater average of the year. These values may explain the two spikes in MR creation observed in weeks 11 and 12. These results help explain the first finding, since it takes more effort to create and close the same number of MRs for the same level of productivity.

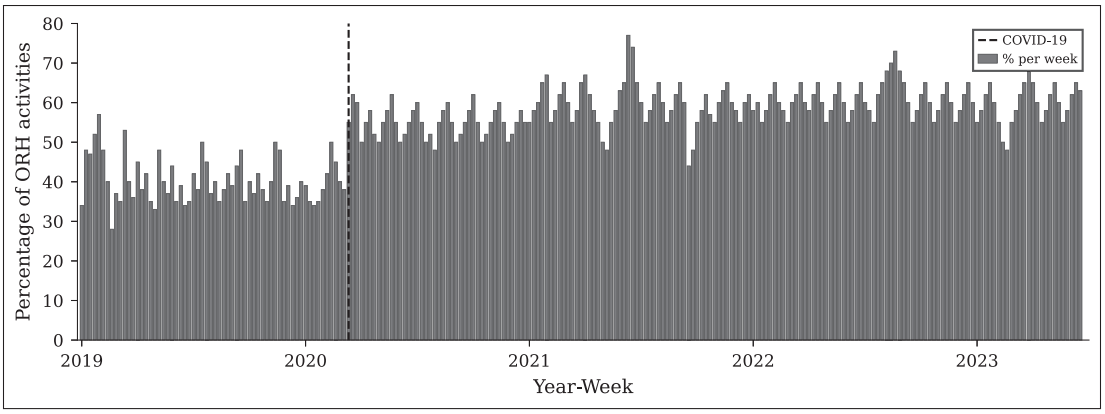


Figure 4.3 The distribution of the percentage of out of regular working hours activities by week before and after COVID-19

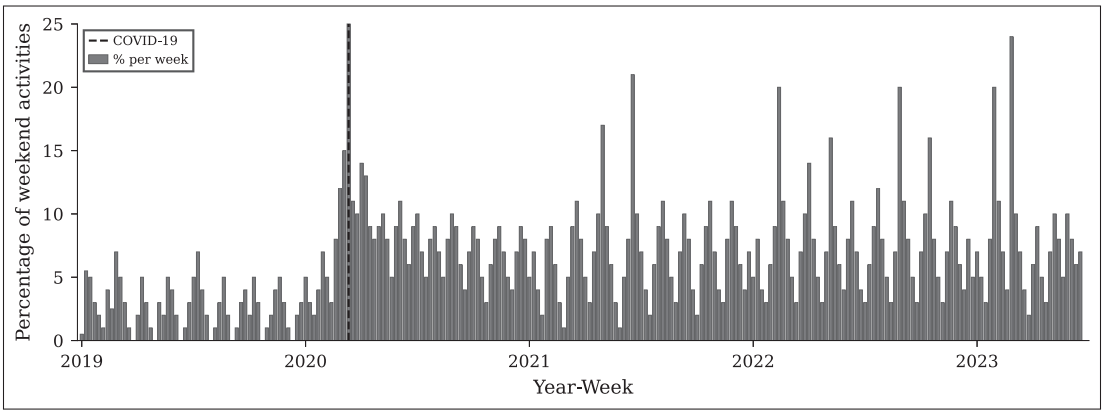


Figure 4.4 The distribution of the percentage of weekend activities by week before and after COVID-19

As shown in table 4.2, activities on MRs outside regular working hours (8 a.m. to 5 p.m.) and on weekends nearly doubled after the pandemic, with up to 70.48% of activities occurring outside regular working hours. As shown in Figure 4.3, there was a notable variation in the percentage of activities done before and after COVID-19. Table 4.2 shows that the mean and median percentages of weekly activities outside regular working hours were 38.50% and 39.45%, respectively, before the pandemic, which increased to 54.56% and 55.91% after the pandemic. Similarly, weekend activities increased significantly, with the median rising from 3.37% to 7.14%, as shown in Figure 4.4. This pattern clearly illustrates the impact of COVID-19 on

developers' temporal autonomy. Working remotely allowed developers more flexibility, not being restricted to the 8 a.m. to 5 p.m. schedule, a trend that has persisted even after the pandemic.

Table 4.2 Out of regular hours (ORH) and weekend statistics activities before and after COVID-19

Period	#MRs	Pattern	Mean %	Median %	Min %	Max %
before COVID-19	8.8k	ORH	38.50	39.45	10.52	58.53
		Weekends	4.49	3.37	0	17.74
after COVID-19	17.94k	ORH	54.56	55.91	34.72	70.48
		Weekends	7.95	7.14	0	25.28

4.4 Internal Process Transitions: OpenShift Migration Case Study

RQ2: Has there been any impact of the process changes, specifically the migration to openshift technology, on different aspect of the organization?

Approach

To study the impact of the migration to OpenShift, we categorize MRs into two groups: those created specifically for the OpenShift migration, targeting a dedicated branch, and other MRs created during the same period of the migration but not related to OpenShift. Considering company's three-week sprint cycle, we study three aspects:

- **Efforts:**We examine the time to complete peer review of MRs, which reflects the effort spent by teams to integrate OpenShift, compared to the effort of completing the other MR group.
- **Change Magnitude:** We assess the MR size, represented by the number of added and deleted lines, to indicate the magnitude of changes introduced to the code. We also compare the number of lines added versus deleted to determine whether developers are primarily adding new code or correcting existing code (by deleting and replacing lines of code).

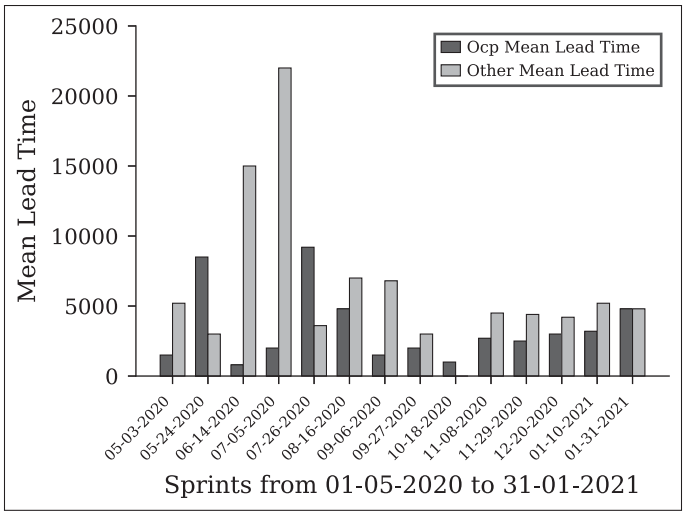


Figure 4.5 Mean time to complete peer review by sprint for OpenShift and other MRs in the migration period

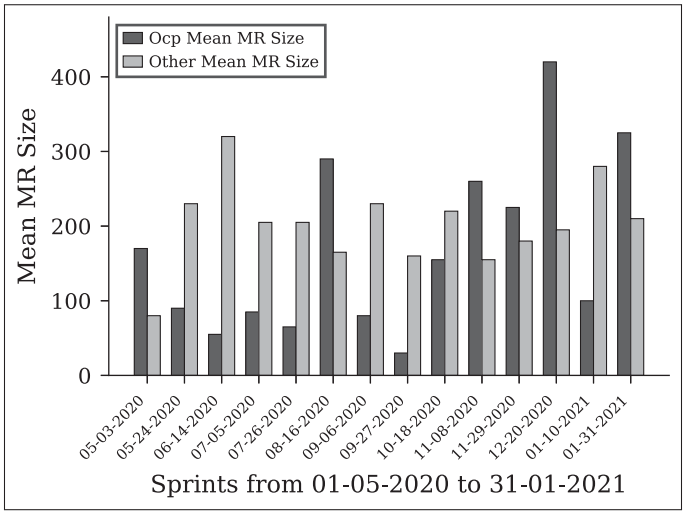


Figure 4.6 Mean size by sprint for OpenShift and other MRs in the migration period

Results

Our results show an inverse relationship between the time to complete peer review of MRs for OpenShift and other MRs during the first eight sprints, as shown in Figure 4.5. When significant effort was spent on OpenShift integration, the time to complete peer review for other MRs increased, suggesting that the focus on OpenShift integration was causing delays in other areas.

Beginning in the eighth sprint, the times to complete peer review for both categories began to converge and vary similarly, eventually converging in the final sprint. This trend suggests that developers initially put extra effort into learning and integrating OpenShift, but as they became more familiar with the technology, the time required for OpenShift MRs converged with the time to complete peer review required for other MRs.

The MR sizes for OpenShift initially were smaller than other MRs but increased as developers gained expertise. As shown in Figure 4.6, until Sprint 8, we observe that MR sizes for OpenShift are smaller compared to others, except for Sprint 6, indicating that developers were slowly integrating new changes into the software. This gradual integration resulted in smaller MR sizes for OpenShift compared to other projects. Starting in Sprint 8, MR sizes for OpenShift begin to increase, becoming larger than other MRs except for Sprint 13. This trend suggests that developers have become more skilled at integrating and reviewing larger chunks of code related to OpenShift.

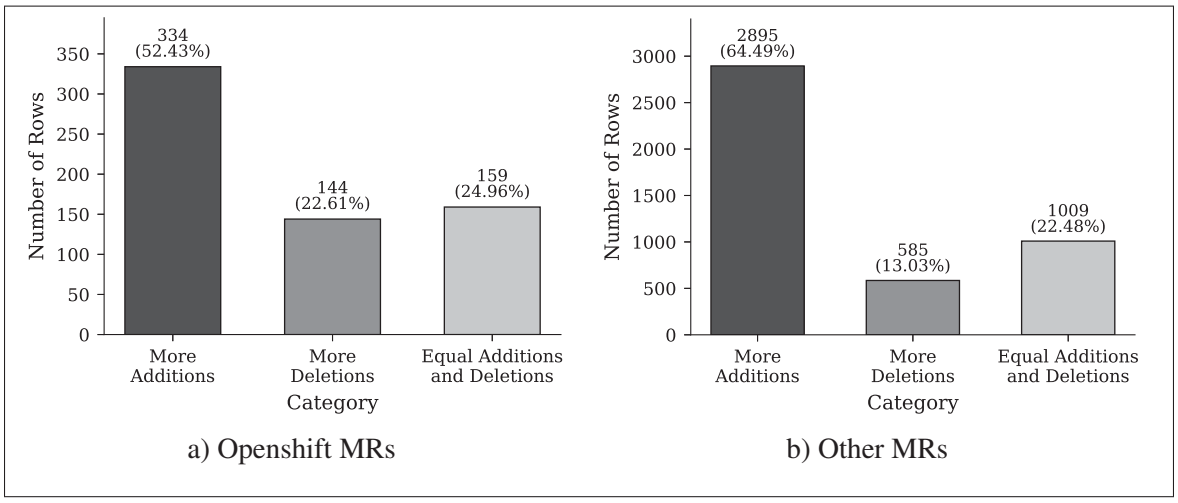


Figure 4.7 Comparison of additions and deletions per MR in Ocp and other MRs

We observe distinct patterns in the nature of MRs for OpenShift and other MRs. For other MRs, 86.97% involve more additions (64.49%) or equal additions and deletions (22.48%), indicating that these MRs are primarily for adding new code or modifying existing code during the review process. Only 13.03% of other MRs involve more deletions than additions. In contrast, 22.61%

of OpenShift MRs involve more deletions than additions, indicating a focus on removing obsolete code. In addition, 52.43% of OpenShift MRs involve more additions, reflecting the integration of new technology and the introduction of new code. In addition, 24.88% of OpenShift MRs involve equal numbers of additions and deletions, indicating significant rework during the peer review process. Such dynamics highlight that modifying the existed code (in 47.57% of OpenShift MRs) can impact code stability, that is considered as an indicator to the product (software) stability Staron *et al.* (2013).

4.5 Organizational Priorities: Branch Traces

RQ3: How are different types of gitlab branches used and managed in an industrial devops environment?

Approach

To analyze the used branch categories using MR data at the studied company, we begin by understanding the company's organization, which operates on a yearly basis divided into seventeen three-week sprints, from Monday to Sunday. GitLab's branch structure remains consistent over the years, with the following categorization:

- **Main branch (Master):** serves as the primary development branch that remains consistent over time, acting as the base for daily development activities and the integration of new features for the latest and greatest product version.
- **Stable branches:** are dedicated to fix bugs and stabilize code quality for a planned release. Stable branches are branched out upon feature complete milestone of a planned release to distinguish between future releases development on master/main and the planned release bug fixes. Different planned releases and their corresponding bug fixes and patch releases are published and released from these stable branches.

- **Temporary branches:** are created for short-term development tasks or special initiatives, such as feature experiments or significant transitions like the migration to OpenShift, these branches are intended for limited use and are merged or discarded once the task is completed.

Note: As shown in Figure 4.9, the number of MRs on the main branch (master) (23K) is always larger than the number of MRs on other branches (1.4K) and stable branches (1.9K), as the daily work is done on the master branch.

Analyzing MRs:

- **Effort and Change Magnitude:** analyze the effort and the size of MRs similarly to the previous RQs.
- **Collaboration Aspects:** analyze the attention received by developers on MRs on different branches, looking at the time to receive the first comment on the MR and the number of discussion notes (messages) required to complete the review process.

By structuring our methodology in this manner, we ensure a comprehensive analysis of the branches, focusing on both the effort and the collaboration required for each category.

Results

Although the MR size distribution of stable and master branches are very similar, we observe that the review process on the stable branch are completed faster than those on the master and other branches. As shown in Figure 4.11, the MR size on the other branch category is larger than on the stable and master branches, which have almost the same distribution. Despite the larger size of MRs on the other and master branches, Figure 4.10 demonstrates that MRs on the stable branch are merged or closed more quickly. Table 4.3 further highlights this point, showing that the mean and median times to complete peer review on the stable branch are three times shorter than those on the master and other branches. This indicates that developers prioritize completing MRs on the stable branch, which is related to the publication of new releases and may indicate the presence of bugs that could impact the delivery of the new version.

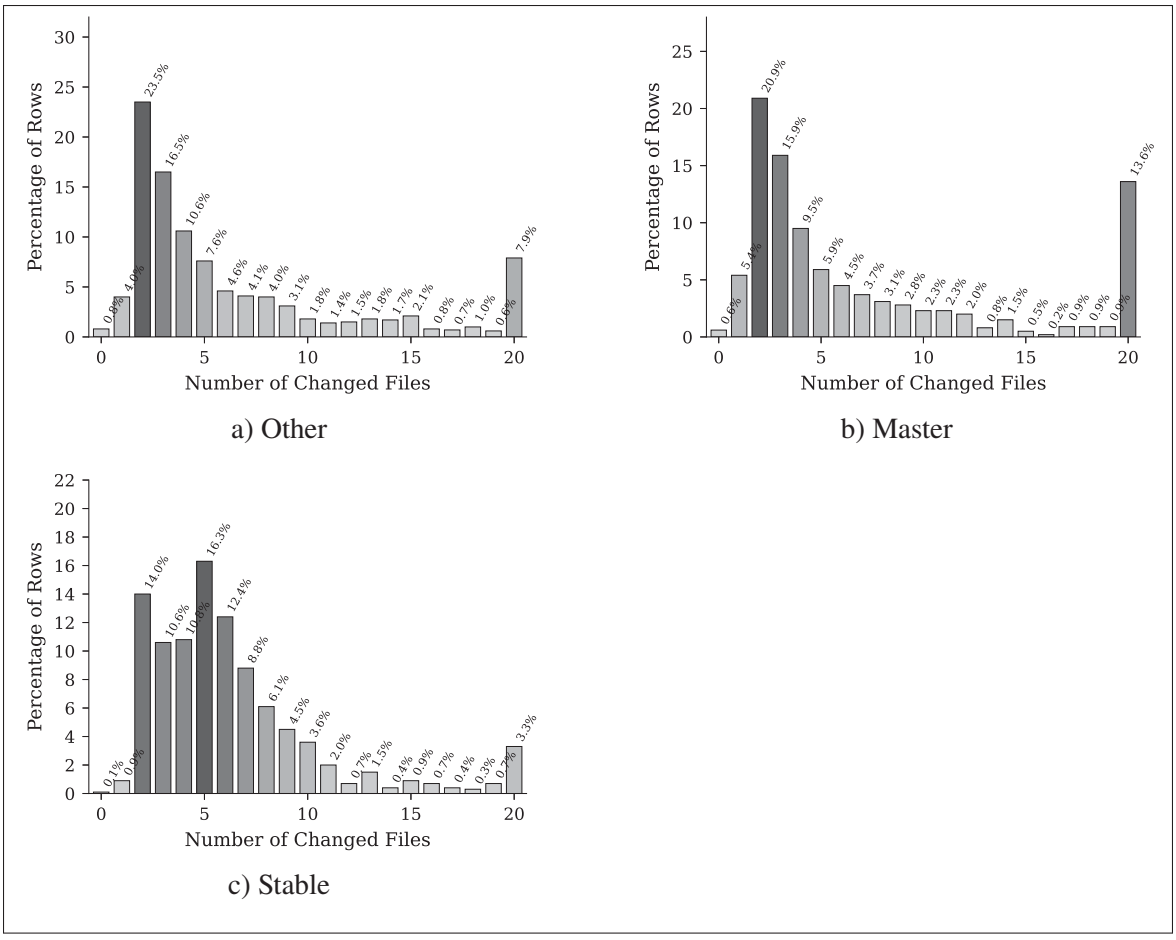


Figure 4.8 Distribution of #notes by branch

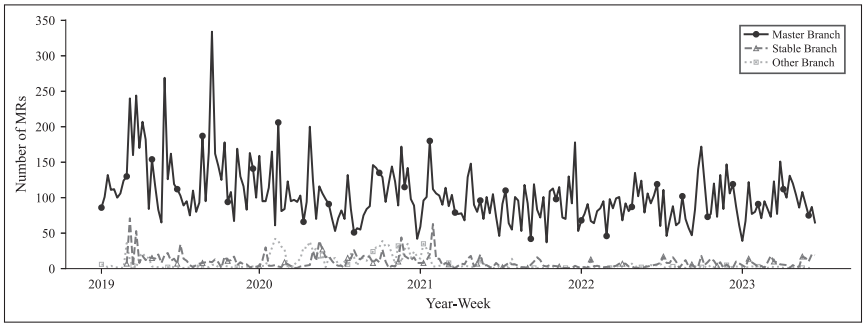


Figure 4.9 Weekly MRs for master, stable, and other branches

As shown in Table 4.3, the stable branch receives the first comment on MRs more quickly, with mean times that are 2.81 and 2.01 times faster than those of the master and other branches,

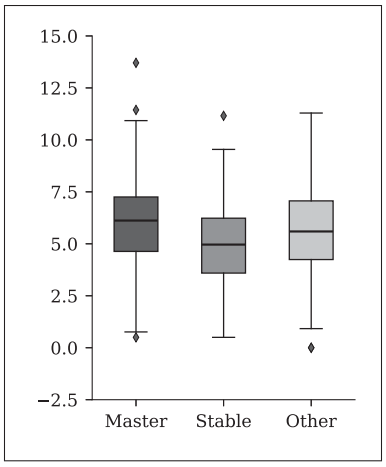


Figure 4.10 Time to complete peer review distribution across branches

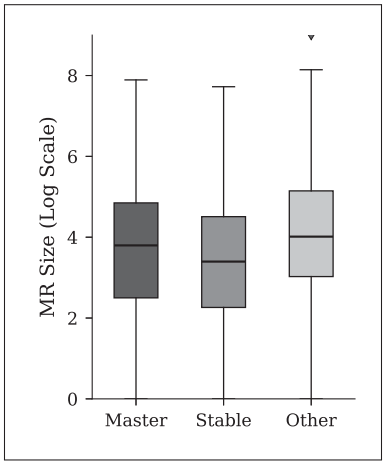


Figure 4.11 MR size distribution across branches

respectively. This is further illustrated in Figure 4.12, which shows the distribution of time to first comment, where the stable branch consistently has shorter times compared to the other branches. Additionally, Figure 4.8 displays the distribution of the number of notes (comments) per MR. For the master and other branches, the largest distributions have 2 notes (23.5% and 20.8%, respectively), which then decrease, indicating fewer MRs with more notes. In contrast, the stable branch shows a different pattern, with the largest group (18.3%) having 5 notes, followed by 12.4% of MRs having 6 notes. This indicates that MRs on the stable branch are more frequently discussed than those on the master and other branches. These observations confirm that MRs on

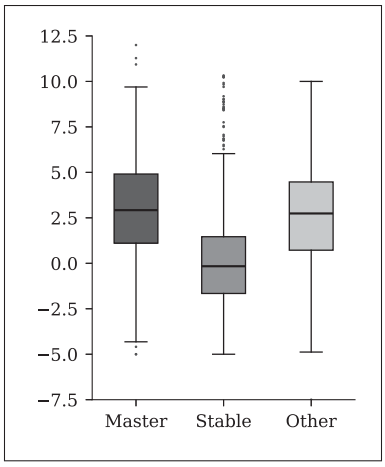


Figure 4.12 Time to first comment distribution across branches

the stable branch receive more attention from team members, who respond quickly and engage in more thorough discussions. This increased attention likely helps to prevent issues or bugs during the delivery process.

Table 4.3 Statistics for master, stable, and other branches

Branch	Length	Time to Complete Peer Review Mean	Time to Complete Peer Review Median	First Comment Mean	First Comment Median
master	23,574	6,606.12	508.36	1,391.01	12.43
stable	1,923	2,988.93	118.62	494.79	0.16
other	1,467	6,481.96	227.75	997.07	10.94

4.6 Peer Review Dynamics: MR Completion Analysis

RQ4: To what extent does the origin of the first comment—bot or human—impact the time to complete peer review?

Approach

Data Preparation

- **Data collection:** Similarly to Hasan *et al.* (2023), we gathered data for each MR, including the discussions surrounding each MR. The collected metrics, indicated by a star (⋆), are detailed in Table 4.1.
- **Label First Comments:** In GitLab, each comment has a "System" parameter indicating whether it was made by a bot or not. We ran an algorithm through the MR data to determine the time between the first comment and the creation of the MR, and label each comment accordingly.
- **Define Comment Categories:** Unlike Hasan *et al.* (2023), who used the BoDeGha tool developed by Golzadeh *et al.* (2021) to classify comments on GitHub, we leveraged the GitLab API, which directly tags system-generated comments. We categorized comments into three types: (1) Bot-generated comments, which provide automated feedback such as "Pipeline failed: check logs for details", helping enforce technical standards; (2) Human comments, offering contextual feedback like "This function could be optimized for readability", which directly impacts code quality; and (3) Bot-labeled but human-driven comments, where system-generated messages, such as "Approved this MR", reflect human decisions despite being tagged as bot comments. Table 4.4 provides further details on these categories.

Data Analysis

- **Time to First Comment Speed:** Analyze the time to receive the first comment compared to the overall time to complete peer review. This step considers the first comment, whether by a human or a bot.

- **Bot vs. Human Comment Comparison:** Compare the speedness of time-to-first-human-comment, time-to-first-extended-human-comment, and time-to-first-bot-comment to understand how bots and human reviewers interact during peer review.
- **Time to Complete Peer Review Analysis:** Explain the time to complete peer review on GitLab involves the following:
  - Construct a regression model to examine the correlation between collected metrics (including time to first comment by bot and human metrics) and the time to complete peer review. Similarly to Zhang *et al.* (2021), we train a mixed-effects linear regression model to explain time to complete peer review. A mixed-effects regression model is appropriate as it accounts for fixed effects (MR characteristics impacting time to complete peer review) and random effects (project-specific variations), enabling generalizable insights while minimizing project structure biases.

    We train a single global model using data from the five Kaloom technical groups, as they yield similar results, allowing for a more concise presentation. Our model achieves an RMSE of 0.65 and MAE of 0.46, which indicate strong performance given the high variance of our dependent variable.
  - Conduct an ANOVA Type-II analysis to identify statistically significant features that explain the variance in time to complete peer review, similarly to prior studies Hasan *et al.* (2023); Kuznetsova, Brockhoff & Christensen (2017). This method evaluates the importance of independent variables without assuming interactions, filtering non-significant features and identifying key predictors using Chi-square (Chisq[1] = 0.05) statistics for robust analysis.
  - Study the impact of key metrics on time to complete peer review using Accumulated Local Effects (ALE) plots, similarly to the approach taken by Khatoonabadi *et al.* (2023). A negative impact indicates shorter time to complete peer review probability, while positive implies longer time to complete peer review probability

[1] Chi-square statistic: measures how well the independent variables explain variability in the dependent variable

Table 4.4 Definition of used categories in this RQ

Term	Definition
Comment	A comment in an MR chat in Gitlab
Human response	A comment given in an MR by a human (but not the MR author)
Bot comment	A comment given in an MR by a bot
Bot but human comment	A suspicious comment, written by a bot but attributable to a human
Human-first MR	An MR whose the first comment is given by a human (not the author of the MR)
Bot-first MR	An MR whose the first comment is given by a bot
Bot-but-human-first MR	An MR whose the first comment is a Bot but human comment
Time to first comment	The time in minute from the MR creation to the first comment
Time to first human comment	The time in minute from the MR creation to the first human comment
Time to first bot comment	The time in minute from the MR creation to the first bot comment

Results

Time to first comment analysis across projects shows generally fast response times, with median values ranging from 4.46 to 23.78 minutes, as shown in Table 4.5. Notably, the FPGA group shows the slowest response times, possibly due to hardware compatibility considerations impacting the interactions on GitLab. This observation suggests the need for deeper investigation into the specific hardware factors impacting response dynamics within this group. Additionally, the median time to complete peer review confirms this finding, showing a significant increase for the FPGA group, exceeding 2600 minutes compared to less than 1200 minutes for the other groups. In other projects, the median time to first comment is faster than 11.44 minutes. For example, in the CP group, the median time to first comment is 9.90 minutes, while the median time to complete peer review is 5304.06 minutes, indicating quick initial feedback compared to the total time required to complete MRs. However, the mean time and standard deviation, which reach 1992.88 and 13955.30 minutes respectively, indicate the presence of outliers that can delay the first comment.

Table 4.5 Time-to-first-comment (TFC) and time to complete peer review (minutes)

Project	Time to Complete Peer Review			TFC		
	Avg	Mdn	Std	Avg	Mdn	Std
mp	3222.95	312.11	17032.33	717.32	11.44	3865.66
cp	5304.06	5304.06	37724.09	980.60	9.90	6132.76
dp	7690.85	1110.44	49330.31	1992.88	10.51	9096.11
fpga	20505.23	2674.39	54782.82	3992.92	23.30	13955.30
pf	6290.53	202.02	31095.99	1102.52	4.46	8470.01

Table 4.6 Proportion of bot-first vs human-first MR comments

Project	% True Bot-First	% Bot-But-Human-First	%True Human- First
mp	42.73%	39.70%	17.58%
cp	59.21%	31.80%	8.99%
dp	66.61%	18.22%	15.17%
fpga	69.77%	17.17%	13.06%
pf	52.55%	39.88%	7.56%

We observe that bot-generated comments dominate the initial interaction in the MR process and are faster compared to human comments. As shown in Table 4.6, up to 69.77% of the initial comments are attributed to bots, with an additional 39.88% of comments initially made by humans but marked as bot. In contrast, human-authored comments are a minority, ranging from 7.56% to 17.58% across the projects studied. Regarding time to have the first comment, analysing Table 4.8, we observe that humans take significantly more time to post a comment than bots. While the median time to first human comment for all groups ranges from 212.49 minutes for MP project to 1611.87 minutes for FPGA project, the global time to first comment ranges from 4.45 minutes for PF to 23.78 minutes for FPGA, which is posted by bots. We observe the same pattern when analysing the average and the standard deviation. This comparison highlights the efficiency of the bots in initiating peer reviews in a timely manner, and highlight their prevalence in accelerating the development workflow and team integration.

As shown in Table 4.7, the time to first human response is the most influential, explaining 93.97% of the variance in time to complete peer review, indicating that faster human intervention to review the changes can statistically impact the overall time to complete peer review of an MR. The number of commits open at the time of the MR, explaining 4.66% of the variance, also plays a crucial role in determining time to complete peer review, likely due to the increased complexity

and larger size of the changes, resulting in more review requirements. When the commenter is the author of the MR, the variance of the time to complete peer review is impacted by 0.60%, suggesting that when authors review their own work, the process is accelerated. The number of previous MRs contributes 0.40% to the time to complete peer review variance, reflecting the importance of team experience in expediting reviews. Description length contributes 0.24% of the variance, highlighting the importance of clear documentation. In addition, time to first bot response, while less influential (0.12%), shows that early bot interactions can help initiate the review process. Our interpretation is confirmed by the ALE findings indicated in Figure 4.13. Overall, these findings highlight that both human and automated responses, along with detailed documentation and reviewer experience, are critical to managing the efficiency of the peer review process.

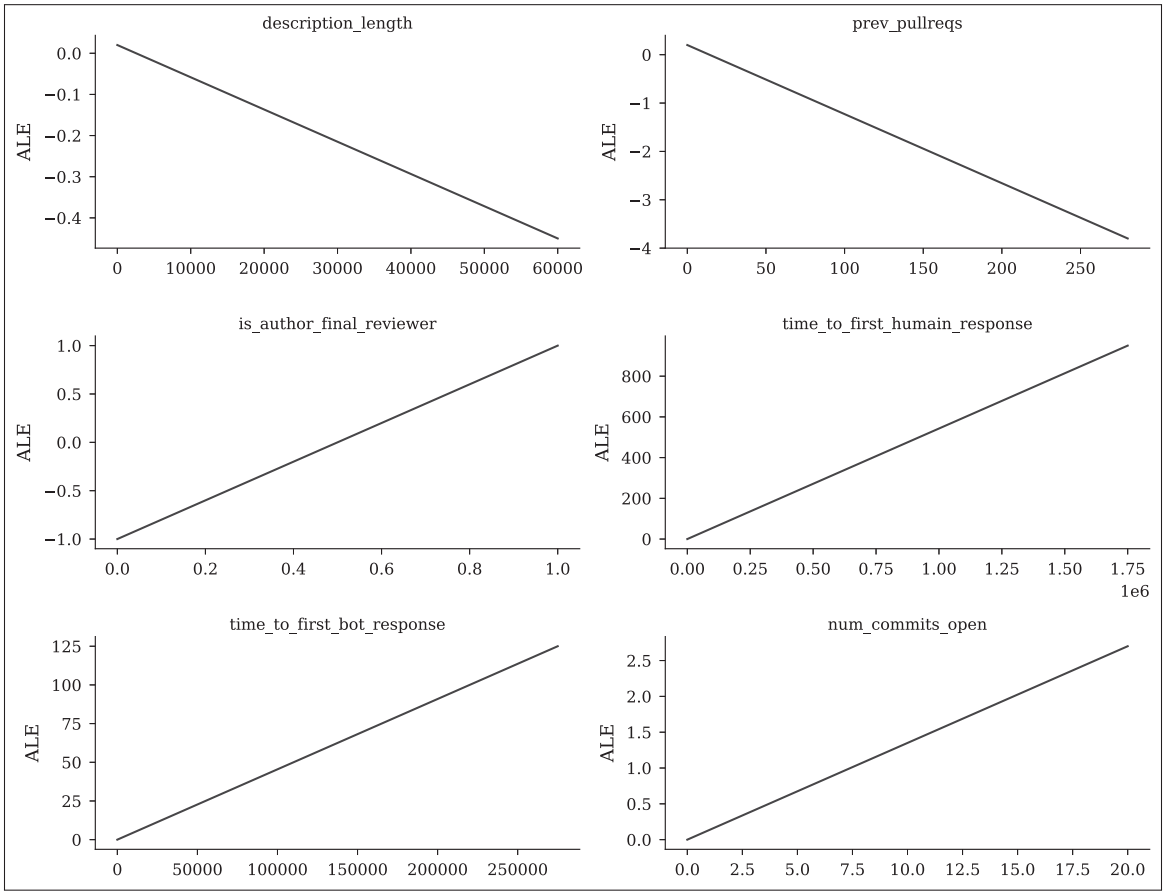


Figure 4.13 Impact of the most important features on time to complete peer review.

Table 4.7 Statistical analysis of features significantly impacting time to complete peer review

Feature	Pr(>Chisq)	Percent
Description Length	5.326583e-03	0.2392008
#Previous MRs	3.103913e-04	0.4006646
Author as Reviewer	9.726386e-06	0.6026754
Time to First Human Comment	0.000000e+00	93.9714120
Time to First Bot Comment	4.705515e-02	0.1214756
#Commits	8.465914e-35	4.6645716

Table 4.8 Statistical analysis of the groups. Time-to-first-human-comment (TFHC), time-to-first-bot-but-human-comment (TFBHC), and time-to-first-human-extended-comment (TFHEC) are given in minutes

Project	TFHC			TFBHC			TFHEC		
	Avg	Mdn	Std	Avg	Mdn	Std	Avg	Mdn	Std
mp	2119.36	212.49	6316.11	690.25	27.81	3084.69	1098.49	42.75	4342.47
cp	4332.32	898.17	16319.31	1491.97	52.14	9686.19	2124.11	78.49	11627.50
dp	6820.87	1357.65	48153.05	2123.68	102.08	10758.76	3865.75	338.37	31395.66
fpga	14111.06	1611.87	40686.73	6381.94	166.59	19846.10	9113.04	921.73	30288.99
pf	4255.92	234.71	13388.22	1161.34	32.30	5099.09	1741.74	40.65	7810.81

4.7 Discussion

The analyses in this chapter support the central claim that MR traces can serve as an observability foundation for DevOps processes. However, this claim should be interpreted carefully. The purpose of the chapter is not to argue that COVID-19, the OpenShift migration, branch management, or bot comments are important only as isolated cases. Rather, these cases are used as representative examples of broader process phenomena: external disruptions, internal technological transitions, organizational prioritization mechanisms, and human–automation interaction patterns. Together, they show that MR traces can reveal how software teams adapt when their normal development conditions change.

The COVID-19 case illustrates this point particularly well. At Kaloom, the transition to remote work occurred rapidly and disrupted a largely office-based work routine. This sudden change created an adaptation period that became visible in MR traces, especially through changes in working-time patterns and activity outside regular hours. However, the same event would not necessarily produce the same signal in every organization. For example, TELUS already had

distributed and partially hybrid work practices before the pandemic. In such a context, the transition to remote work may have introduced less behavioral novelty in the MR workflow. Therefore, the absence of a strong COVID-19 signal in another company would not imply that the organization was unaffected; it may simply indicate that the pre-event baseline was already closer to the post-event working mode. MR traces should therefore be understood as instruments for detecting departures from routine behavior, rather than as universal measures of organizational disruption.

A similar interpretation applies to the OpenShift migration. The migration is not important only because it concerns OpenShift specifically, but because it represents an internal technological transition that temporarily changed development and review conditions. In this case, MR traces captured a period of instability followed by gradual stabilization. This suggests that MR-based observability can help identify adaptation periods during internal process or technology changes. However, such signals are easier to detect when the transition is sufficiently concentrated in time and scope. More gradual or distributed changes may be harder to observe without additional contextual information about when and where the transition occurred.

The branch-prioritization analysis further shows that MR traces can reveal organizational priorities embedded in workflow behavior. Faster handling of stable-branch MRs indicates that some changes receive prioritized treatment, likely because of their relationship to release management. Nevertheless, MR traces expose the presence of this prioritization more directly than its exact cause. Such patterns may result from explicit branch policies, release-management rules, maintainer authority, or informal coordination practices. For this reason, MR traces are especially useful for detecting where prioritization occurs, while complementary sources such as branch policies, release documentation, or practitioner interviews may be needed to explain why it occurs.

The analysis of bot-initiated and human-initiated comments also illustrates the value and limits of MR traces. In the studied context, bot comments mainly corresponded to automated feedback such as CI reports, linting, or templated messages, making the distinction between bots and

humans relatively clear. The results show that bots accelerate initial responses, while human comments remain more strongly associated with review progression. However, this distinction may become less stable as LLM-based review agents are integrated into DevOps workflows. Future studies may need to distinguish between different types of automated comments, such as mechanical feedback, summarization, and substantive LLM-generated review feedback, rather than treating all bot comments as a single category.

Overall, these findings refine rather than weaken the observability claim. MR traces do not provide a complete representation of the software process, since some review and coordination activities occur outside the MR platform through direct messages, meetings, or informal discussions. However, they provide a rich, longitudinal, and low-intrusion record of process behavior at scale. The four analyses presented in this chapter show that the same type of data can reveal meaningful signals across different forms of process variation: external disruption, internal transition, organizational prioritization, and human–automation interaction. This establishes MR traces as a defensible empirical foundation for the remaining chapters, where peer review effort is analyzed not as an isolated review outcome, but as part of a broader DevOps process.

4.8 Implications

4.8.1 Implications for our industrial partners

Beyond the scope of this chapter, these results suggest practical value for our industrial partners and for software teams more broadly. The systematic analysis of MR traces can help teams assess how development activity responds to organizational or environmental change, identify prioritization patterns in workflow management, and better understand the complementary roles of automation and human collaboration in peer review. More specifically, industrial partners can leverage MR-based analytics to monitor the impact of process transitions, such as tooling migrations or the adoption of new CI/CD infrastructure, on review behavior and workflow stability. They can also use branch-level indicators to make release-prioritization practices empirically observable and to validate that stable branches are receiving the expected degree

of review attention. Finally, automated first-comment detection can help teams calibrate the balance between bot-driven and human-driven review activity, ensuring that bots accelerate review initiation without substituting for the substantive review progression that still depends on human reviewers.

4.8.2 Implications for researchers

For researchers, the findings of this chapter position MR traces as a methodological instrument that extends well beyond the study of peer review activity alone. By showing that MR data make environmental disruptions, internal technological transitions, organizational priorities, and automation–human interaction dynamics empirically observable, this chapter broadens the empirical scope in which MR-based studies can be legitimately conducted. Researchers investigating DevOps, software process improvement, or socio-technical change in software organizations can therefore treat MR traces as a longitudinal, low-intrusion observation channel that complements traditional sources such as surveys or interviews, while avoiding recall bias and retrospective reconstruction. The four observability dimensions examined in this chapter (environmental, internal-technological, organizational, and interactional) can also be reused as a framing template for future empirical studies that aim to analyze software processes from MR data rather than from review activity in isolation.

The results also raise methodological considerations for future empirical work. First, because MR traces reflect both technical activity and contextual change, empirical studies that aggregate MRs over long periods without accounting for such contextual shifts (e.g., remote-work transitions, infrastructure migrations, or release-management policies) may conflate distinct regimes of behavior and produce results that are harder to interpret or replicate. Explicitly segmenting longitudinal MR datasets around documented contextual events is therefore a research practice that this chapter supports empirically. Second, the distinction between bot-initiated and human-initiated review activity suggests that automated comments should be systematically identified and reported in peer review studies, as their presence can inflate surface-level activity metrics without reflecting substantive reviewer engagement. Failing to do so risks overestimating human

reviewer involvement in automated pipelines and may bias comparative analyses across projects or organizations with different automation practices.

4.9 Threats to Validity

Internal Validity Threats: These threats are related to how we calculated our metrics. We ensured that our metrics are defined according to established literature and that our implementation scripts are accurate. However, despite these precautions, there is always a possibility of implementation errors. We took steps to verify the correctness of our data and the accuracy of our scripts, but we cannot completely rule out the occurrence of errors.

External Validity Threats: These threats concern the generalizability of our findings beyond the studied company. Our conclusions are drawn from a single industrial setting, and DevOps practices can differ significantly across organizations due to factors such as company size, industry domain, development culture, and tooling preferences. Furthermore, different companies experience unique events—such as organizational restructurings, policy shifts, or external disruptions—that may impact their DevOps processes in ways that our MR-based analysis may or may not fully capture. For instance, major process shifts (e.g., adoption of new CI/CD pipelines, teams restructuring, or external regulatory changes) could alter MR dynamics in ways that are not directly observable in our dataset. Additionally, the metrics we used may need adaptation or extension to analyze different organizational contexts effectively. Other peer review mechanisms, such as Gerrit or GitHub pull requests, have structural differences in review workflows, reviewer roles, and decision-making processes compared to GitLab's MR system. This means that while our results provide valuable insights into GitLab-based workflows, they may not fully translate to companies using other platforms.

4.10 Conclusion

In this chapter, we examined whether data extracted from MR traces can support the analysis of software process phenomena beyond the peer review process itself. Through an empirical

study of 26.7k MRs collected from an industrial networking software company, we showed that peer review traces capture meaningful signals related to multiple dimensions of DevOps process behavior, including environmental disruption, internal technological transition, organizational prioritization, and local review interaction dynamics. These results establish that MR traces constitute a rich and expressive source of process information rather than a narrow source of review metrics alone.

This foundation motivates the remainder of the thesis. Having established that MR traces provide a valid and expressive source of broader process information, the next chapter builds on this basis to examine peer review effort more directly through a time-independent perspective centered on rework.

CHAPTER 5

UNDERSTANDING THE AMOUNT OF CHANGES REQUIRED FOR MERGE REQUEST ACCEPTANCE: AN EMPIRICAL STUDY

Samah Kansab[1] , Mohammed Sayagh[1] , Francis Bordeleau[1] , Ali Tizghadam[2]

[1] Department of Software Engineering and IT, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[2] TELUS, 25 York St, Toronto, Ontario, Canada M5J 2V5

Paper Published in the 16th IEEE International Conference on Software Engineering and Service Science on December 2025

Preamble

This chapter proposes the *amount of changes required to complete peer review* — the number of lines added and deleted after MR creation — as a time-independent indicator of peer review effort. Using four open-source GitLab projects, we find that post-submission change affects up to 73% of MRs (18% on industrial data), is not correlated with time to complete peer review or reviewer count, and can be predicted at MR creation with an AUC of up to 0.88. Complexity, author experience, and textual features are the most consistent drivers of high-change MRs.

This chapter is adapted from the paper published at ICSESS 2025. The thesis version revises the introduction, conclusion, and overall framing to align the chapter with the objectives of this thesis.

5.1 Introduction

As established in the previous chapter, MR traces provide a valuable empirical basis for observing DevOps process behavior. The analyses in that chapter showed that review activity can reveal changes in the development process by capturing how practitioners interact with MRs and how reviews progress under different conditions. However, because this activity also reflects the effort required to complete peer review, it must be interpreted through appropriate metrics.

Building on this need for clearer effort measurement, this chapter focuses on the effort required to move an MR from submission to acceptance.

Measuring peer review effort is important because it reflects the work, coordination, and adjustment needed after a change is submitted. However, effort is difficult to capture with a single indicator. Much of the existing literature has approached peer review effort through temporal measures, such as time to complete peer review, time to first comment, and related latency indicators (Chouchen *et al.*, 2023; Zhang *et al.*, 2022; Maddila *et al.*, 2023; Huang *et al.*, 2020; Hasan *et al.*, 2023; Thongtanunam *et al.*, 2017b; Fan *et al.*, 2018; Saini & Britto, 2021; Islam *et al.*, 2022). These measures are useful for understanding review duration and responsiveness, but they do not isolate the work generated by the review process itself. An MR may remain open for a long time while requiring little corrective work, whereas another may be revised extensively in a short time because it receives immediate attention.

This chapter therefore studies peer review effort through a time-independent indicator: the *amount of changes required to complete peer review*. We define this measure as the sum of lines added and removed across all commits made after MR creation. Conceptually, it captures the volume of post-submission modification performed before acceptance and therefore reflects the corrective or adjustment work that follows the initial submission. This perspective complements latency-based indicators by focusing on what changes during review, rather than how long the review takes.

The closest related studies have used the number of review iterations or patchsets as indicators of review rework (Wang *et al.*, 2019, 2021). However, iteration counts do not capture the magnitude of the revisions they contain: one iteration may introduce only a few changed lines, whereas another may involve substantial modification. By measuring the actual amount of post-submission change, this chapter provides a more direct view of review-induced work.

This chapter pursues three objectives. First, it characterizes how frequently and how substantially MRs are modified after submission. Second, it examines whether post-submission change captures an effort dimension distinct from time-based and participation-based indicators. Third,

it develops predictive models to identify, at MR creation time, MRs that are likely to require substantial post-submission work. This perspective is valuable both analytically and practically: it broadens the empirical understanding of peer review effort beyond time and may support earlier identification of review-intensive MRs.

Using four open-source GitLab projects (Section 3.4), we find that post-submission change is common, affecting up to 73% of MRs before acceptance. The measure is not correlated with time to complete peer review or with the number of involved practitioners, indicating that it captures a distinct dimension of peer review effort. Predictive models achieve an AUC of up to 0.88 when classifying MRs according to the amount of changes required to complete peer review. Feature analysis further shows that complexity-related metrics are the most consistent drivers of high-change MRs, while experience, text, and historical features also contribute to explaining this form of effort.

The main contributions of this chapter are:

- the definition of the *amount of changes required to complete peer review* as a time-independent indicator of peer review effort, measured as the sum of lines added and removed across post-submission commits;
- empirical evidence that this measure captures an effort dimension distinct from time-based and participation-based indicators, and therefore complements existing views of peer review effort;
- predictive models that identify, at MR creation time, MRs likely to require substantial post-submission work, together with an analysis of the features that drive this form of effort.

5.2 Study Design

In this chapter, we examine peer review effort through a time-independent perspective based on the amount of changes required to complete peer review. The objective is not only to characterize how much post-submission change MRs typically undergo, but also to determine whether this dimension of effort is distinct from conventional time-based indicators and whether it can be

identified early. To achieve this objective, we design an empirical study structured around three research questions. Together, these questions allow us to characterize the distribution of post-submission change, evaluate the predictive performance of machine learning models, and identify the factors most strongly associated with high-change MRs.

The study relies on MR data collected from four open-source GitLab projects, as described in Section 3.4. Based on this collected data, we extract a set of derived metrics to characterize different dimensions of the MR, its context, and its review process. Because the three research questions pursue different objectives, they do not rely on exactly the same analytical procedures. Instead, each question is addressed using the subset of metrics and analysis techniques most appropriate to its purpose.

5.2.1 Research Questions

The chapter is guided by the following three research questions:

RQ1 What is the amount of changes required to complete peer review? This question explores how much code is typically changed before an MR is accepted, and how this effort correlates with known review process metrics like time to complete peer review and reviewer participation (Chouchen *et al.*, 2023; Thongtanunam *et al.*, 2017b). While prior studies have focused on timing and involvement to assess review quality (Wang *et al.*, 2019; Maddila *et al.*, 2023), the extent of changes required during review remains underexplored. Since more changes often imply more effort from both authors and reviewers, this metric offers a valuable lens for understanding review challenges. We analyze its distribution and assess whether it aligns with or differs from traditional review metrics.

RQ2 How effectively can machine learning models classify MRs according to the amount of changes required to complete peer review? This question investigates whether MRs can be classified according to their amount of post-submission change before acceptance, with the objective of identifying a predictive model suitable for early detection of high-change cases. Being able to anticipate MRs likely to require substantial revision may support

better planning and earlier intervention during the review process. As in prior predictive studies of peer review behavior, this requires robust models with sufficiently strong performance and stability. This question therefore evaluates the effectiveness of different machine learning models for this classification task.

RQ3 Which factors most strongly impact the amount of changes required to complete peer review? This question examines the factors associated with higher post-submission change. Identifying such factors is useful both analytically and practically. Analytically, it helps explain what drives this form of effort. Practically, it may support the definition of better review practices and earlier risk awareness for MRs likely to require substantial revision. This question therefore studies the importance and impact of the different feature dimensions used in the predictive setting.

5.2.2 Metric Extraction

To support these three research questions, we collect a total of 33 metrics spanning seven dimensions: *Text*, *Collaboration*, *Experience*, *History*, *Branching*, *Date*, and *Complexity*. These dimensions are intended to capture complementary aspects of MRs, including their textual properties, collaboration patterns, author and reviewer background, historical activity, branch context, temporal context, and change-related complexity.

Some of these metrics are directly available from the GitLab platform, whereas others are derived from raw MR and commit data. Together, they provide a multi-dimensional view of the conditions in which post-submission change occurs. The full list of metrics, their definitions, and the associated hypotheses are presented in Table 5.1. Since the three research questions do not pursue the same analytical goal, the role of these metrics differs across the chapter. Their specific use is therefore detailed in the corresponding approach subsection of each research question.

Table 5.1 List of collected metrics

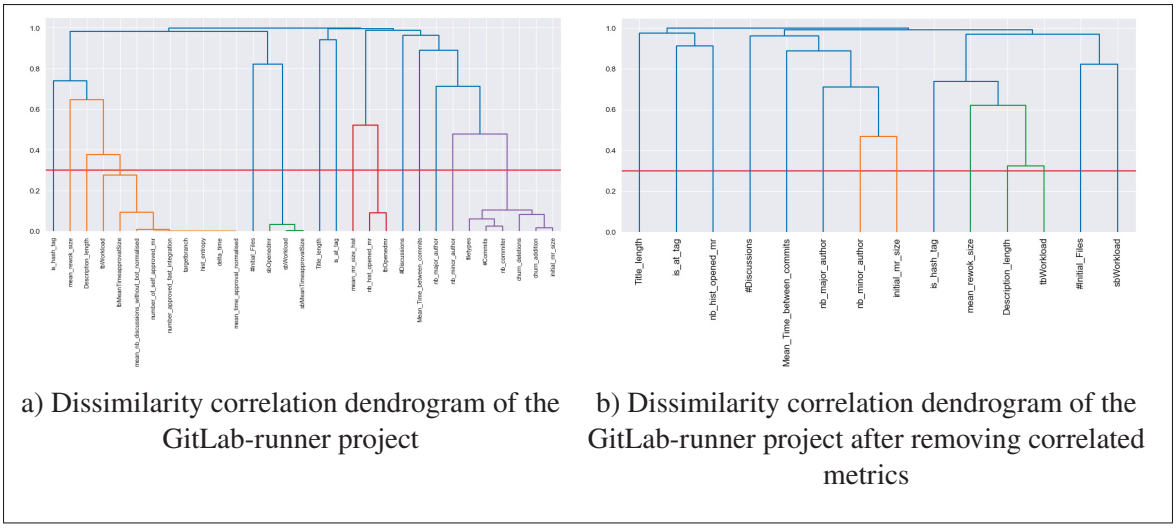
Metric	Description	Dimension	Hypothesis
Hashtag in Title/Description	True if the description or title contains a hashtag	Text	Text-related metrics may provide additional information about the MR, helping practitioners identify MRs likely requiring large amounts of changes.
At-Tag in Title/Description	True if the description or title contains an at-tag		
Description Length	Length of the description of the MR		
Title Length	Length of the title of the MR		
Number of Commits	Total number of commits	Collaboration	A higher number of commits may indicate incremental changes, which can increase the number of changes required for accepting MRs.
Number of Unique Committers	Number of unique committers involved		
Mean Time Between Commits	Average time between commits		
Number of Discussions	Number of comments left by authors		
Minor Authors	Number of authors who created the MR and contributed less than 5% of changes in historical MRs	Experience	Experience may affect the quality of submitted code, potentially requiring more changes for acceptance.
Major Authors	Number of authors who created the MR and contributed more than 5% of changes in historical MRs		
Approved Fast Integrations	Number of historical MRs where changes exceeded 200 per hour	History	A higher number of fast integrations may indicate efficient collaboration and a streamlined review process, potentially affecting the number of required changes.
Historical Opened MRs	Number of historical MRs that remain open		
MRs Without Discussion	Number of historical MRs without discussion		
Mean Approval Time (Normalized)	Mean approval time normalized by the number of changes in historical MRs		
Mean Discussions Without Bots (Normalized)	Mean number of discussions without bots normalized by number of changes in historical MRs		
Self-Approved MRs	Number of historical MRs where the creator was the integrator		
Mean MR Size (Historical)	Mean size of historical MRs (initial size before creation + accepted changes)		
Mean Changes per MR (Historical)	Mean number of changes for historical MRs		
Source Branch Workload	Number of prior MRs on the same source branch	Branching	A higher number of prior MRs on the same source branch may indicate more active development history, potentially increasing required changes.
Source Branch In Progress	Number of prior MRs still open on the same source branch		
Source Branch Delay	Mean approval time normalized by changes of prior MRs on the same source branch		
Target Branch Workload	Number of MRs with the same target branch		
Target Branch In Progress	Number of prior MRs still open on the same target branch		
Target Branch Delay	Mean approval time normalized by changes of prior MRs on the same target branch		
Delta Time	Time between the creation of the project and the MR	Date	Longer delta times may indicate more mature projects, potentially affecting the review process and number of required changes.
MR Creation Date	Date of creation of the MR		
Number of File Types	Number of file types included in the MR	Complexity	We suggest that more complex code and changes lead to a higher number of required changes for accepting MRs.
Number of Initial Files	Number of initial files included in an MR		
Initial MR Size	Initial size of the MR (added + deleted lines before creation)		
Churn Addition	Number of initial lines added		
Churn Deletion	Number of initial lines deleted		
Change Entropy	Distribution of changes among initial files in prior MRs		

5.2.3 Correlation and Redundancy Analysis

Because collinearity between features can affect the interpretability and stability of machine learning models (Tantithamthavorn, McIntosh, Hassan & Matsumoto, 2016b; Cito, Dillig, Kim, Murali & Chandra, 2021), we perform correlation and redundancy analysis before building the predictive models. Following prior work (Jiarpakdee, Tantithamthavorn & Hassan, 2019; Tantithamthavorn & Hassan, 2018), this step is applied separately to each dataset in order to reduce feature collinearity while preserving representative information.

First, we compute Spearman correlations between metrics and use a threshold of 70% to identify strongly correlated feature pairs (McIntosh *et al.*, 2014; Lee *et al.*, 2020). We then apply dissimilarity dendrogram clustering and retain one metric from each correlated group when the dissimilarity is below 0.3. For example, in the GitLab-runner project, we retain the number of historical opened MRs, which captures similar information to its branch-specific counterpart (Figure 5.1).

Second, we perform redundancy analysis based on R^2 in order to identify features that can be largely explained by the others. Following prior work, we exclude features whose variance can be explained at a level greater than 90% (Tantithamthavorn, Hassan & Matsumoto, 2018; Lee *et al.*, 2020). This two-step process helps reduce feature overlap and improves the interpretability of the predictive analysis.



a) Dissimilarity correlation dendrogram of the GitLab-runner project

b) Dissimilarity correlation dendrogram of the GitLab-runner project after removing correlated metrics

Figure 5.1 Dissimilarity dendrograms used in the correlation analysis

5.2.4 Effort Labeling and Noise Reduction

To support the predictive analyses, we transform the effort measure into a binary outcome by discretizing MRs into *large-change* and *small-change* classes. This transformation is performed using median-based discretization, which is appropriate here because the objective is to distinguish MRs requiring relatively high or low amounts of post-submission change.

Because discretization may introduce noise that can affect model performance and interpretation (Rajbahadur, Wang, Kamei & Hassan, 2019), we apply an additional noise reduction step. Following prior work, we optimize this process using the Area Under the ROC Curve (AUC), which is considered a stable and reliable evaluation metric for binary classification (Huang & Ling, 2005; Mossman, 1994; Halimu, Kasem & Newaz, 2019). Importantly, noise removal is applied only to the training set, while the test and validation sets remain unchanged in order to preserve the integrity of the evaluation procedure. The median thresholds used for discretization are computed from the full dataset and are reported in Table 5.2.

Table 5.2 Summary of changes required for accepting MRs across studied projects. The MR counts reported here correspond to the dataset snapshot used for the predictive analyses of this chapter, after filtering out MRs with missing change histories; the canonical per-project profile of the open-source dataset is reported in Section 3.4

Project	Number of MRs	Min	Median	Mean	Max
Omnibus	7,217	0	14	95	5,695
GitLab Runner	4,417	0	25	197	18,492
Inkscape	6,003	0	20	189	12,727
Gitaly	6,000	0	44	199	12,685

5.2.5 Analysis Strategy

The analysis strategy varies according to the objective of each research question, but follows a common overall logic. First, we operationalize peer review effort through the amount of post-submission change required before acceptance, defined in this chapter as the sum of lines added and removed across commits made after MR creation. This measure serves as the dependent variable of the study.

For RQ1, we use descriptive and comparative analyses to characterize the distribution of this measure and examine its relationship with selected review-related indicators. For RQ2, we formulate a classification task that separates merge requests according to their amount of post-submission change and compare the predictive performance of several machine learning models. For RQ3, we analyze the importance and effect of the different feature dimensions in order to identify the factors most strongly associated with higher-change merge requests.

The methodological details of these analyses are presented in the corresponding approach section of each research question. This organization keeps the study design coherent at the chapter level while allowing each research question to use the most suitable analytical procedure for its objective.

5.3 Understanding Amount of Changes Required to Complete Peer Review

RQ1: What is the amount of changes required to complete peer review?

Approach

In this study, we define the amount of changes required to complete peer review as the total number of added and deleted lines (i.e., additions + deletions) in all commits made after the MR is created. We only consider commits that are committed after the MR creation, regardless of their authored date, since authored commits are not visible until they are committed. At the time of MR creation—our threshold—only committed changes are available and included in our analysis.

We analyze the distribution of the amount of changes across MRs and examine its correlation with key review metrics such as time to complete peer review, number of reviewers, committers, commenters, and total comments.

We use the Spearman correlation between different metrics Spearman (1961), using the following interpretation:

- *Very High Correlation*, if $0.9 < |r| \leq 1$
- *High Correlation*, if $0.7 < |r| \leq 0.9$
- *Moderate Correlation*, if $0.5 < |r| \leq 0.7$
- *Low Correlation*, if $0.3 < |r| \leq 0.5$
- *Negligible Correlation*, if $0 \leq |r| \leq 0.3$

Note that we focus on the correlation between the amount of changes required to complete peer review and two key review metrics: peer time to complete peer review and number of reviewers, as these have been widely studied in prior work. A strong correlation may suggest that findings related to time to complete peer review and reviewer count could also apply to the amount of changes. Otherwise, if the correlation is weak, it implies that prior conclusions may not generalize, and a separate analysis of the amount of changes is necessary.

Results

66%, 66%, 68% and 73% of the analyzed MRs require an amount of changes during the peer review process for Omnibus, GitLab Runner, Gitaly, and Inkscape, respectively.. The density distribution in Figure 5.2 highlights a concentration of MRs with an amount of changes in the 1–200 range, with an average of 38% across all projects. Among the MRs that require changes, up to 28% involve at least 200 lines of code being modified during the review process. In addition, we observe that certain MRs exhibit more than 1,000 changed lines across all projects. For instance, within the Omnibus (OB) project, out of 7,217 MRs, 4,739 require changes after submission. Among these, 2,861 involve a change size between 1–200 lines, while 1,878 require at least 200 lines of changes. Notably, 76 MRs within this group show an amount of changes exceeding 1,000 lines—reaching up to 5,695, as shown in Table 1. These findings underscore the importance of managing the amount of changes required to complete peer review, as it not only increases the size and complexity of an MR but also raises the risk of errors, redundancy, and communication challenges. It can further strain contributors, especially when working under tight deadlines.

The number of additions and deletions are highly correlated across all studied projects, with Spearman correlation coefficients exceeding 0.9. Specifically, we observe values of 0.97, 0.96, 0.96, and 0.92 for the Omnibus, GitLab Runner, Inkscape, and Gitaly projects, respectively—indicating a strong association between added and deleted lines of code. These results suggest that most changes within MRs involve modifications to existing code, rather than purely new additions or removals. This dynamic emphasizes that the amount of changes

required to complete peer review may affect code stability, which is often considered an indicator of overall software product stability Staron *et al.* (2013).

We observe low and negligible correlations between the amount of changes required to complete peer review and both peer time to complete peer review and the number of people involved in the MR. The Spearman correlation values between peer time to complete peer review and the amount of changes are 44%, 42%, 47%, and 48% for the Omnibus, GitLab Runner, Inkscape, and Gitaly projects, respectively. This suggests that the time taken to complete a peer review is not strongly related to the size of the code changes made during the process.

In contrast, the correlation between the amount of changes and the number of comments is notably higher—71%, 67%, 64%, and 67% for Omnibus, GitLab Runner, Inkscape, and Gitaly, respectively—indicating that developers often need to review, comment on, and revise code within the same limited time frame.

The Spearman correlation between the number of contributors and the amount of changes is low—20%, 19%, 29%, and 19% for the four projects—suggesting no meaningful relationship between how many people are involved and the size of the changes required. For example, in the Inkscape project, an MR with 1,200 lines of changes involves only three contributors, while a change-free MR involves ten. Similarly, in GitLab Runner, an MR with about 7,500 lines of changes involves six reviewers, whereas one with 2,500 changes involves 13 contributors. These observations reinforce the conclusion that the amount of changes required to complete peer review is largely independent of the number of collaborators involved.

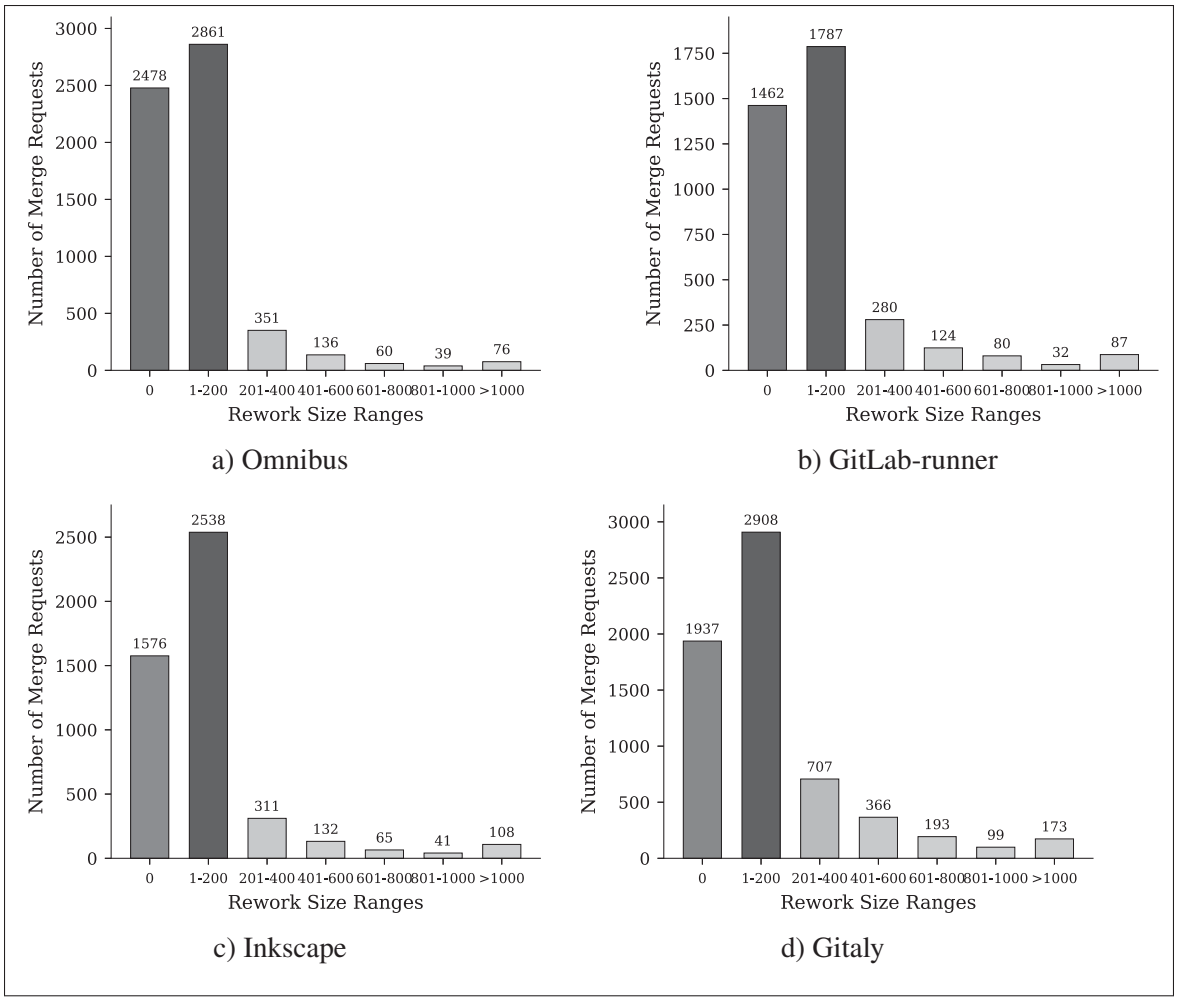


Figure 5.2 Distribution of MRs by the amount of changes required to complete peer review in four projects

5.4 Predicting Amount of Changes Required to Complete Peer Review

RQ2: How effective are machine learning models at classifying the amount of changes required to complete peer review?

Approach

To evaluate the effectiveness of machine learning (ML) models in classifying the amount of changes required to complete peer review, we conduct a comparative analysis involving four

distinct classifiers: K nearest neighbor **(KNN)**, Classification and Regression Trees **(CART)**, Random Forest **(RF)**, and Logistic Regression **(LR)**, which have been used in previous studies (Rajbahadur, Wang, Kamei & Hassan, 2017; Rajbahadur *et al.*, 2019; Jiarpakdee *et al.*, 2019). Our approach is structured as follows:

- **Model Training:** To explain the amount of changes required to complete peer review, we evaluate four different classifiers K nearest neighbor, random forest, decision tree, and logistic regression. We use the 100 out-of-sample bootstrap technique similar to prior studies (Rajbahadur *et al.*, 2017; Tantithamthavorn *et al.*, 2016b) for obtaining statistically robust results and conclusions. We generate the samples for both training and testing sets in each iteration, while we use the validation set for all the bootstraps samples. We ensure that we have 10 data points for each feature to train the model (e.g. if we have 20 features so we need at least 200 MRs).
- **Performance Evaluation:** To evaluate our model, we divided the data set into training and test sets, with 80% of the data in the training set and 20% of the data in the validation set. The validation set consists of the most recent MRs in terms of creation date. For evaluation metrics, we use the commonly used metrics in the literature (Rajbahadur *et al.*, 2019; Gong, Rajbahadur, Hassan & Jiang, 2021; Tantithamthavorn *et al.*, 2018; Zimmermann & Nagappan, 2008), which are Accuracy (ACC), Precision (PRC), Recall (RCL), Brier Score (BS), Area Under Curve (AUC), F-measure (FM), and Mathew's Correlation Coefficient (MCC). We generate 100 bootstrap samples for each metric, resulting in 100 values for each sample for the and 100 values for each sample for the validation.
- **Classifier Ranking:** We rank all ML regressors performance for both test and validation using Scott-Knott Effect Size Difference (ESD) Test [1], which clusters distributions into statistically distinct ranks.
- **Statistical Difference:** To identify subtle differences in performance between two classifiers when their performance is closely matched, we use the Anova test with a significance level of p-value = 0.05. This statistical test compares the distributions of a given metric. If the resulting p-value is less than 0.05, we interpret it as indicating a significant difference

[1] https://cran.r-project.org/web/packages/ScottKnottESD/

between the distributions. Conversely, if the p-value is greater than 0.05, we conclude that there is no statistically significant difference between them.

Next, we assess the importance of each metric dimension in the best-performing model by iteratively removing one dimension at a time and retraining the model on 100 bootstrap samples. We compare the median performance (across bootstraps) of each reduced model to that of the full-feature model to determine the impact of each dimension. Statistical significance is evaluated using ANOVA to identify meaningful performance differences, providing insight into each dimension's contribution to classification accuracy.

- **Model Training:** By iteratively removing one dimension at a time, we train the model on 100 bootstrap samples. The comparative analysis of the model performance, taking into account the removal of metric dimensions, allows the ranking of the importance of the dimensions.
- **Impact:** we consider the impact of removing one dimension on the baseline model (model with all features) as the median of 100 bootstraps of the model without dimension on the median of the 100 bootstraps of the baseline model.
- **Statistical Difference:** similarly to the previous step, we use ANOVA test to measure the statistical distribution significance between the performance metrics, providing valuable insight into the contribution of each dimension in the classification accuracy.

Results

Across all projects studied, the Random Forest (RF) classifier shows the best performance on 67% of the measured metrics when evaluated on test datasets. As shown in the Figure 5.3, particularly on the GR, GT, and Ink projects, random forest outperforms other classifiers in all metrics except recall where Classification and Regression Trees (CART) occurs as the top-1 classifier. However, for the OB project, RF outperforms other classifiers only in precision, with CART outperforming in all other metrics. Interestingly, when RF doesn't take the top 1 position, its performance metrics closely align with CART, suggesting a subtle difference between these two classifiers.

The good performance of our models are consistent on the validation datasets, whereas RF achieves the best performance in 96% of the measured metrics, as shown in Figure 5.4. This consistency confirms the reliability of our model in classifying unseen MRs based on the amount of changes required to complete peer review. In our comparative analysis of classifiers, RF classifier achieves the first rank with an AUC range from 0.84 and 0.88, except for the recall metric for the GT and OB projects, where CART performs best. However, in both case studies, RF outperforms the other classifiers in all other metrics. To evaluate the significance of performance differences between RF and other classifiers, we perform ANOVA test. The results presented in table 5.3 indicate a significant difference in the distribution of 100 bootstrapped performances between RF and other classifiers for all case studies, underscoring that RF consistently achieves significantly better results compared to the other classifiers.

Our analysis shows that excluding any metric dimension results in a significant decrease in performance across all projects studied, suggesting that all the used dimensions are relevant in the classification of MRs based on their amount of changes. Figure 5.5 illustrates that using the random forest classifier identified as the best model for amount of changes classification, the removal of various metric dimensions results in a consistent decrease in performance across all studied projects. When examining the impact, as shown in Figure 5.6, we find that removing any metric dimension negatively impacts all the metrics, except for the GR project. In the case of the GR project, only the removal of the complexity dimension results in a negative impact on the model, whereas the removal of other dimensions yields a positive impact.

Table 5.3 Anova test of RF with other classifiers (p-value)

	Clf	***1-MSE***	***Precision***	***Recall***	***AUC***	***MCC***	***ACC***	***F1***
OB	CART	**1.2e-20**	**1.4e-04**	**1.3e-45**	**1.0e-20**	**5.4e-20**	**1.2e-20**	**4.8e-25**
	LR	**5.1e-78**	**1.2e-60**	**4.1e-64**	**1.2e-77**	**1.7e-77**	**5.1e-78**	**3.0e-89**
	KNN	**1.1e-193**	**2.6e-173**	**2.4e-138**	**4.2e-194**	**3.4e-194**	**1.1e-193**	**1.2e-178**
GR	CART	**5.6e-20**	**8.7e-38**	**5.3e-05**	**1.4e-18**	**2.2e-20**	**5.6e-20**	**2.8e-15**
	LR	**4.9e-92**	**8.6e-60**	**2.4e-87**	**1.9e-94**	**1.1e-90**	**4.9e-92**	**1.8e-91**
	KNN	**1.2e-157**	**7.9e-146**	**6.2e-116**	**2.0e-157**	**4.2e-158**	**1.2e-157**	**1.1e-149**
GT	CART	**3.1e-23**	**7.3e-45**	**5.9e-04**	**2.7e-25**	**2.7e-24**	**3.1e-23**	**4.4e-19**
	LR	**2.4e-110**	**1.4e-56**	**3.4e-114**	**6.4e-110**	**5.0e-109**	**2.4e-110**	**2.1e-108**
	KNN	**1.5e-172**	**1.9e-155**	**1.6e-117**	**1.5e-173**	**4.1e-173**	**1.5e-172**	**2.6e-162**
Ink	CART	**3.1e-30**	**1.4e-62**	**5.3e-07**	**3.0e-24**	**5.7e-31**	**3.1e-30**	**1.6e-23**
	LR	**1.3e-149**	**5.6e-97**	**3.2e-142**	**1.6e-153**	**2.4e-145**	**1.3e-149**	**2.0e-150**
	KNN	**7.9e-183**	**1.8e-163**	**2.3e-124**	**3.3e-180**	**6.4e-183**	**7.9e-183**	**1.7e-169**

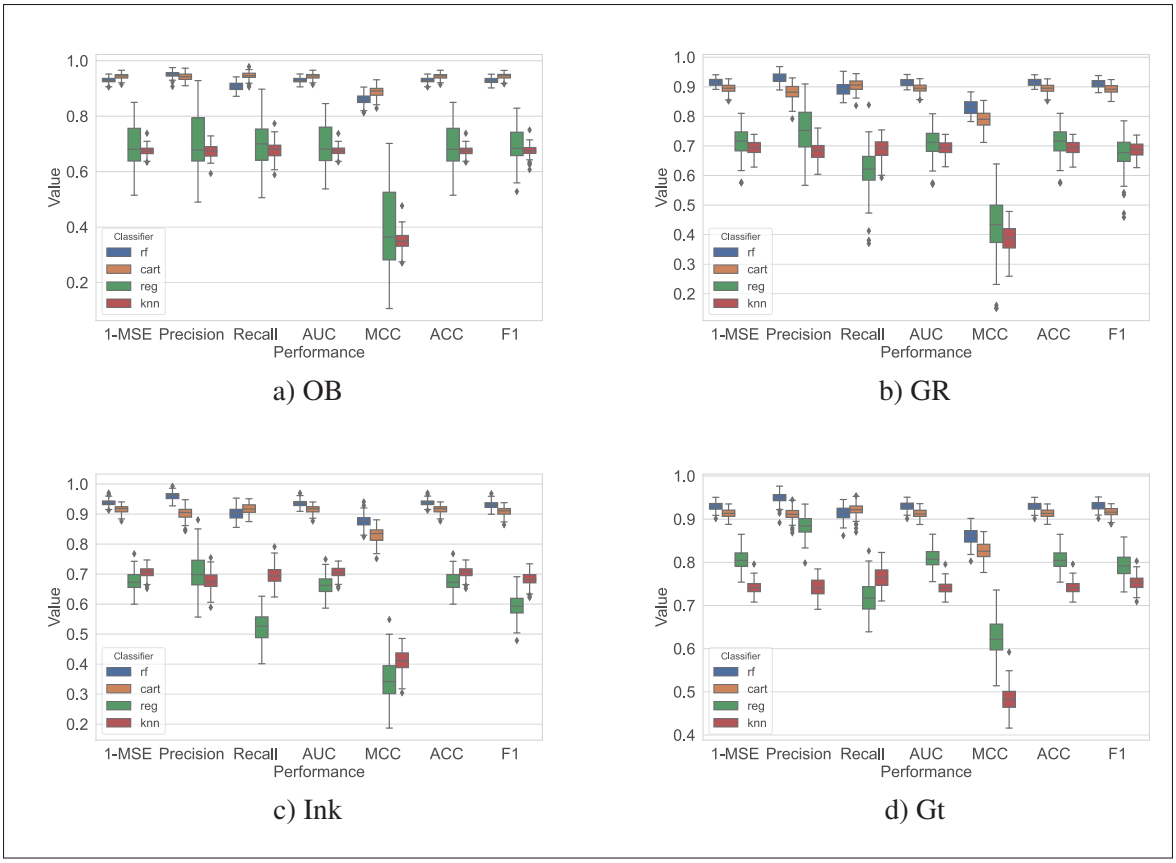


Figure 5.3 Comparative multi box plots for performance distribution on test datasets

RQ3: What are the most important features that influence the amount of changes required to complete peer review?

Approach

In this step, we use Random Forest Classifier (RF), that shows the best performance in RQ2, to study the factors that impact the peer review metrics. To do so, we follow these steps:

- **Perform permutation analysis:** this technique is used to assess the statistical significance of feature importance by comparing actual model performance with random permutations in the dataset independent metrics, providing a robust method for identifying key factors that explain the dependent variable. We apply permutation analysis on each of 100 generated bootstraps using validation dataset to assess the feature importance.

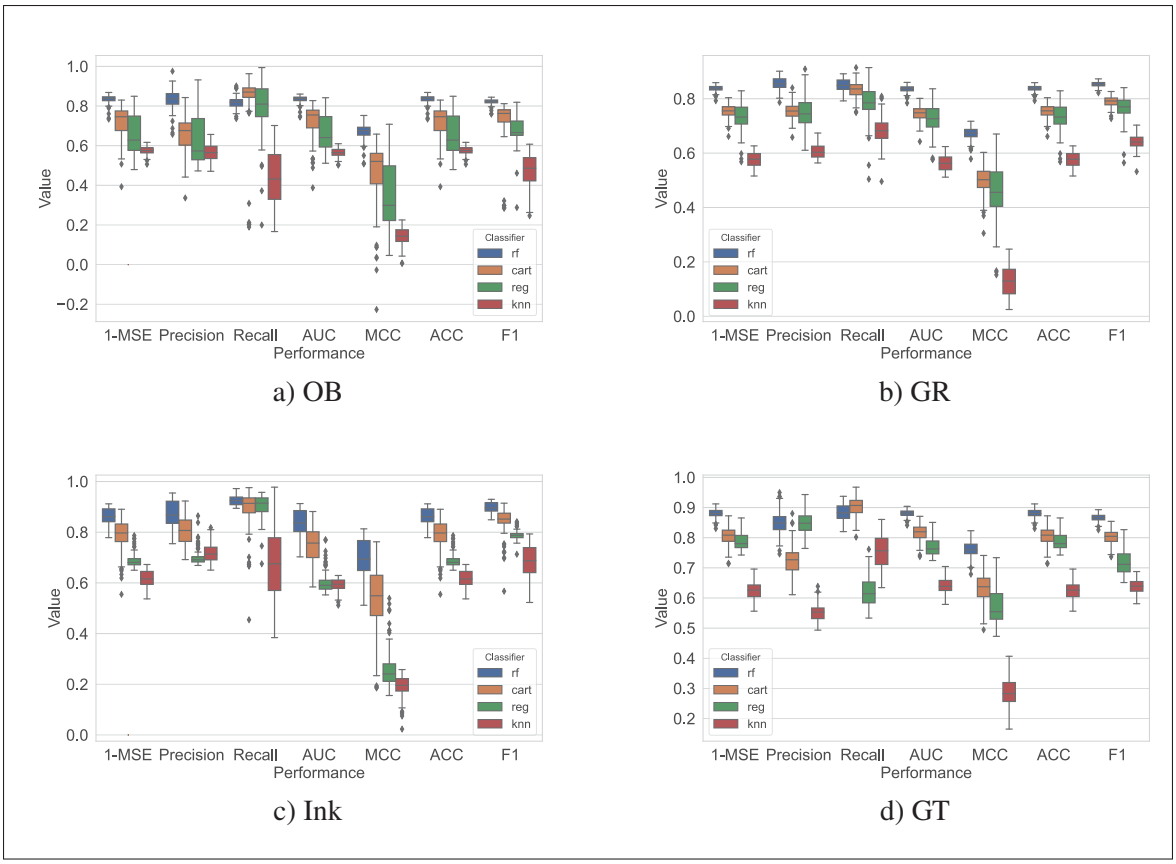


Figure 5.4 Comparative multi box plots for performance distribution on validation datasets

- **Rank the features across all 100 bootstrap samples:** Similarly to the performance ranking, we rank features by their importance while accounting for variability of ranks across different samples using Scott-Knott Effect Size Difference (ESD) Test.
- **Analyze associations between metrics and the amount of changes:** Using Accumulated Local Effects (ALE) plots (Figure 5.8), we explore the relationships between independent variables and the amount of changes required to complete peer review, in order to understand their impact on the classification outcome. Following the approach of Khatoonabadi *et al.* (2023), we exclude feature values exceeding the 99th percentile in each project to reduce the influence of outliers. Each subplot in Figures 5.8 shows the marginal impact of a specific metric (x-axis) on the predicted probability of requiring a large number of changes (y-axis). A decreasing ALE value suggests a higher likelihood of a small amount of changes, while an

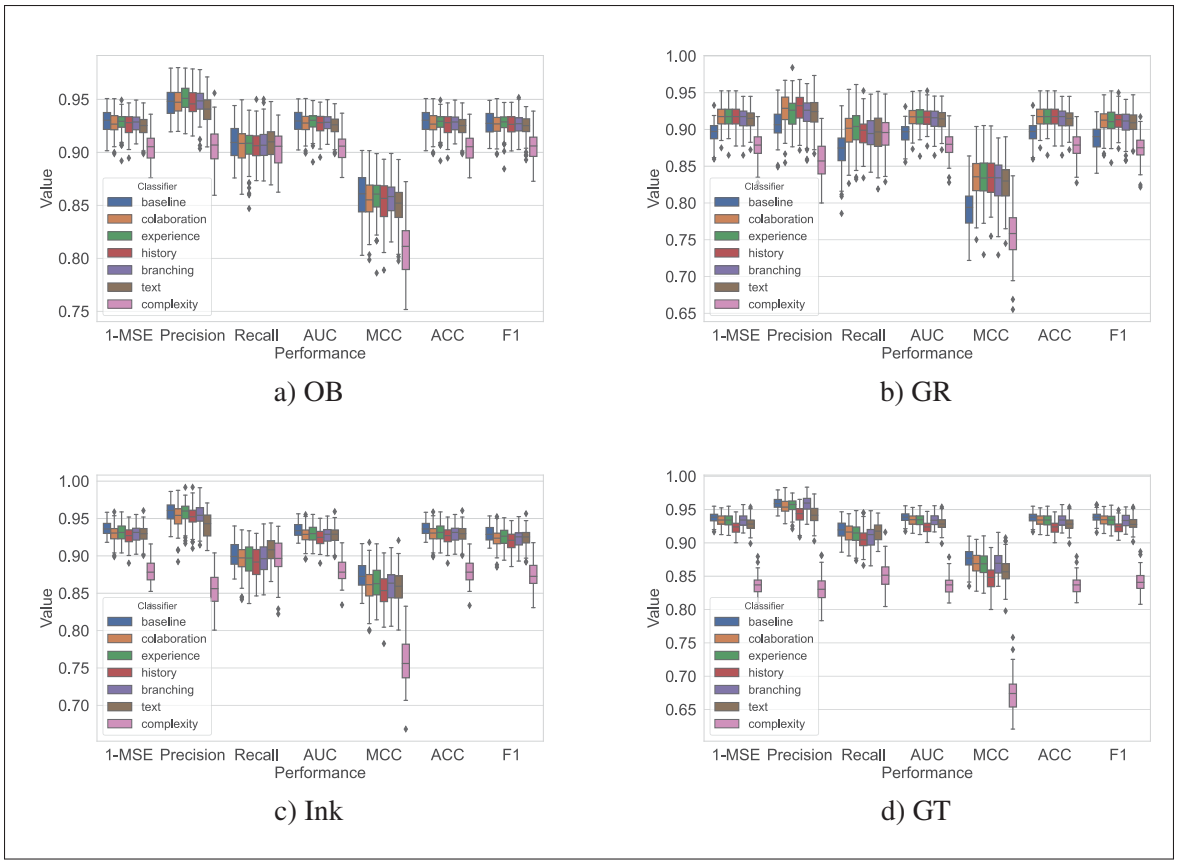


Figure 5.5 Comparative multi box plots for performance distribution when removing metric dimensions

increasing ALE value indicates a higher probability of a large amount of changes for the given MR.

Results

Our analysis shows that complexity metrics are the two most important factors influencing the amount of changes required to complete peer review. As illustrated in Figure 5.7, for all the studied projects, the number of initial files, followed by the initial size of the MR, rank as the first and second most important features, respectively. As shown in Figure 5.8, the increasing number of initial files implies a larger amount of changes. We suggest that a higher number of initial files in an MR indicates a more complex change during peer review, especially if those

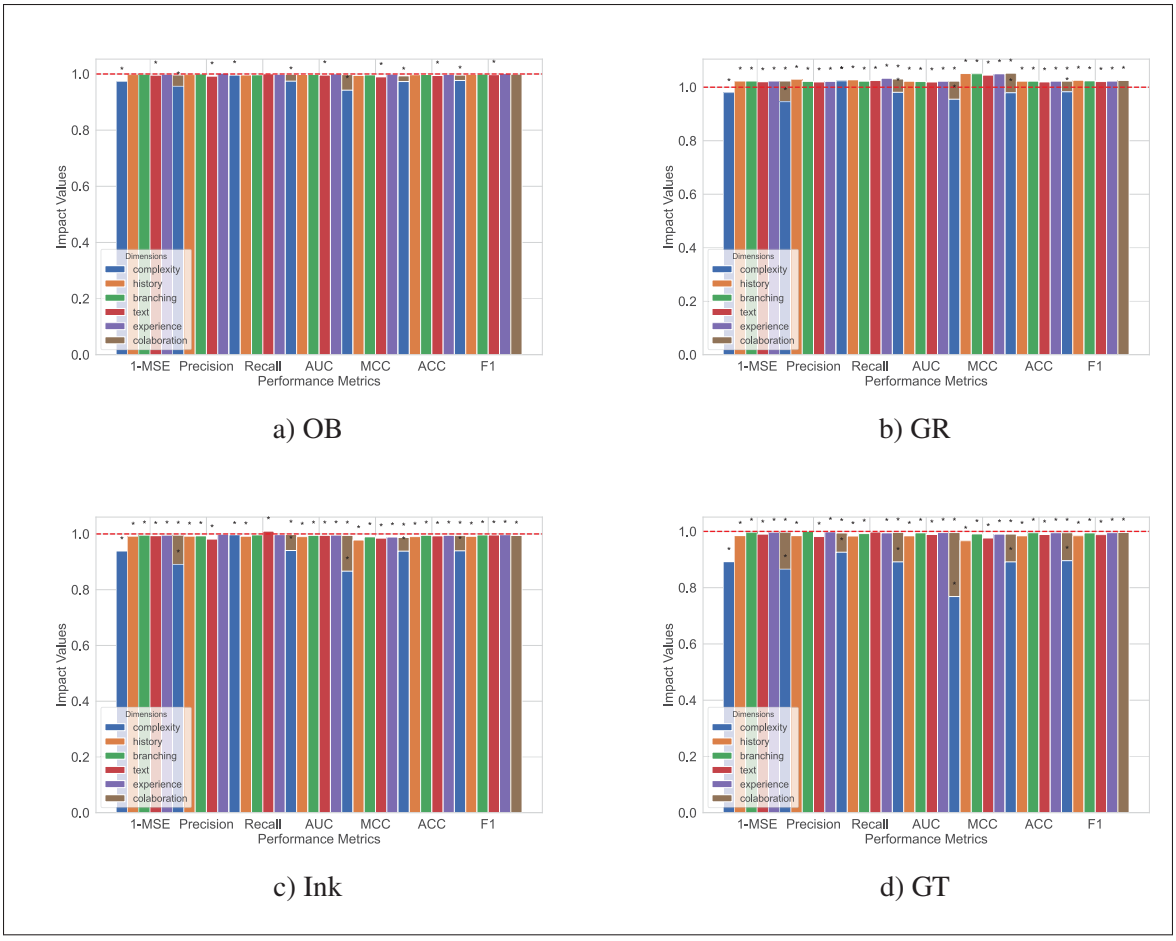


Figure 5.6 Comparative multi bar chart for performance distribution when removing metric dimensions

Note: The bars marked with an asterisk (*) indicate significant impact using the Anova test. The horizontal line helps to differentiate the positive and negative impacts.

files are interdependent. Interdependency in this context indicates that a change in one file requires updates in all related files, thereby increasing the amount of changes. For example, changing an attribute in a configuration file requires updates in all associated files, increasing the number of lines of code to change. At the same time, except the GR project, we observe that a larger initial size of an MR tends to correspond to a smaller amount of changes required during the peer review process. This pattern suggests that in some cases, MRs with large initial sizes indicate that necessary changes are being made early on, before the MR is created. Examples of such purposes include adding documentation, changing branches, or improving modularity by splitting existing code into multiple files. These actions involve significant amount of the lines

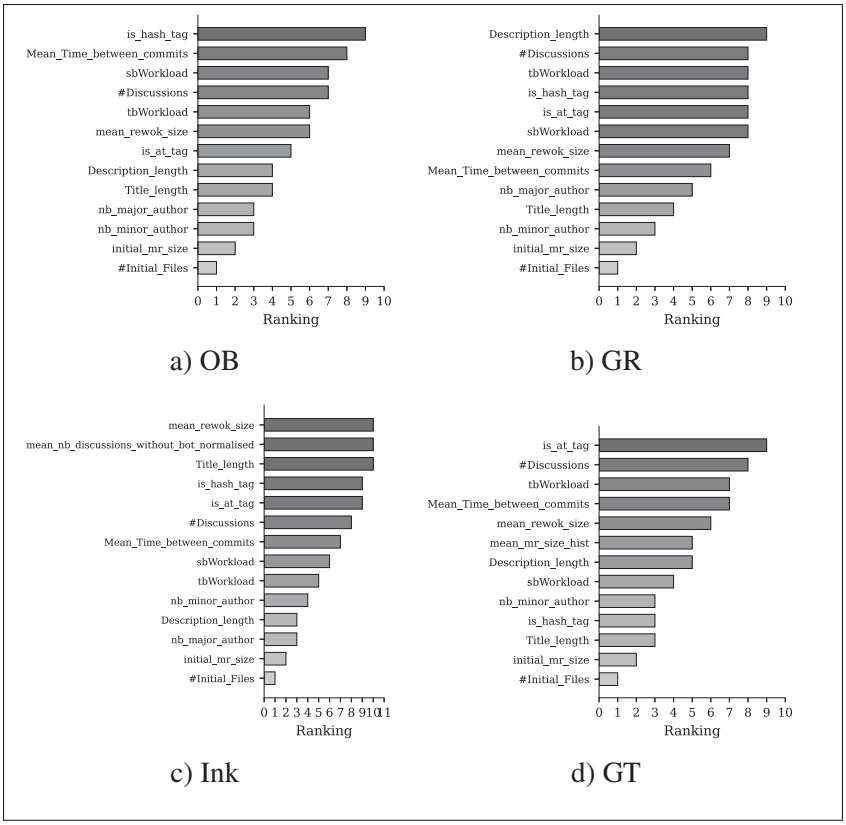


Figure 5.7 SK-ESD feature importance ranks

of code modified before the MR is created, but typically do not require code changes during the review. Conversely, smaller initial MRs may indicate the addition of new configurations or features. Such an introduction often requires extensive code changes for integration into the main codebase, resulting in the need for more review and larger amount of changes.

Author experience metrics also have a significant impact on amount of changes required to complete peer review, consistently ranking in the top-3 or top-4 in 3/4 of studied projects. Figure 5.7 shows that the number of major authors ranks as the third most important feature for OB and Ink projects, and fifth for GR projects. As shown in Figure 5.8, the impact of this metric is negative for all projects, suggesting that having more major authors, who typically have experience with the project, is associated with smaller amount of changes. We interpret this by experienced authors contribute code that requires fewer adjustments or revisions during the review process. In the other hand, we observe that the number of minor authors ranks third

in importance for GR, GT, and OB projects, and fourth for the Ink project. In all the studied projects, the impact of the number of minor authors on the classification of amount of changes is consistently negative. This implies that the presence of minor authors tends to result in smaller amount of changes. We suggest that collaboration between major and minor authors may represent effective mentorship and knowledge transfer within the team. Experienced authors guiding less experienced contributors can result in code that require less changes, highlighting the positive impact of such collaborative dynamics on peer review.

Text-related metrics consistently rank among the top five most important features across all projects. As shown in Figure 5.7, description length ranks third in Ink, fourth in OB, and fifth in GT. Figure 5.8 shows its consistently positive impact, suggesting that longer descriptions are associated with larger code changes. This likely reflects submissions requiring major revisions—such as new features or services—before merging. For instance, in GT, an MR introducing distributed reads had a 3997-character description and 2995 lines of code changed[2], while another with no description simply fixed a typo with zero changes[3].

Title length is also important, ranking in the top five for GR, OB, and GT. However, its impact varies—positive in half the projects and negative in the rest—suggesting that titling practices differ by context.

Tag-related metrics also appear in the top five for OB and GT. The presence of an "@ tag" generally correlates with fewer changes in 3 out of 4 projects, possibly because tagged individuals focus the review on their area of responsibility. In contrast, hashtags show project-specific effects. Their meaning varies—from technical tags like #bugfix or #feature to communication tags like #teamA—which may explain their inconsistent impact on change size.

Historical metrics also show an impact on amount of changes in two projects. Figure 5.7 shows that for the Ink project, the workload (number of MRs) on the source and target branches impacts the amount of changes. The positive impact of these metrics suggests that an increased workload

[2] https://gitlab.com/gitlab-org/gitaly/-/merge_requests/2738

[3] https://gitlab.com/gitlab-org/gitaly/-/merge_requests/4422

can potentially reduce code quality by having more to-do tasks, which results in large amount of changes. For the GT project, the average amount of changes required to complete peer review for historical MRs ranks in the top five. We suggest that the skill level of the individuals involved in the project requires more changes to produce the required quality of code. In addition, we suggest that this observation may be related to the nature of the project, which involves substantial MRs such as new feature additions or updates, that can require more changes.

5.5 Discussion

The contribution of this chapter is to position the amount of changes required to complete peer review as a distinct dimension of peer review effort, rather than as a replacement for time-based measures. Time to complete peer review captures the duration of the review process, but it is affected by waiting time, prioritization, reviewer availability, and organizational constraints. By contrast, post-submission change captures the observable modifications made after an MR is submitted. This makes it useful for studying the work generated during review, but it should be interpreted as a measure of the scale of review-induced rework, not as a complete measure of its cognitive or technical difficulty.

This distinction is important because the same amount of changed lines may represent different levels of effort. A small semantic correction can require more reasoning than a large formatting change, while a large mechanical refactoring may inflate the metric without corresponding to high review difficulty. The amount of post-submission change should therefore be understood as one effort dimension among others. Its value lies in complementing temporal indicators by capturing what changed during review, while time-based measures capture how long the review process took.

The interpretation of this measure may also evolve with the increasing use of AI and LLM-based coding assistants. The definition of effort used in this chapter assumes that post-submission changes largely reflect work performed by practitioners in response to review. However, when an LLM generates part of the revision, the same change volume may correspond to less direct

human implementation effort. Conversely, AI support may encourage reviewers or authors to make larger revisions because producing them becomes easier. In both cases, the relationship between changed lines and human effort becomes less stable. This does not invalidate the measure, but it means that future studies should interpret post-submission change in light of the development context, especially whether revisions are mainly human-written, AI-assisted, or AI-generated.

This point is particularly important when comparing open-source and industrial settings. In the studied open-source projects, post-submission change affects up to 73% of MRs, whereas in the industrial datasets only 18% at industrial projects required observable post-submission changes. This difference does not necessarily mean that industrial MRs require less effort. Some review-related work may happen before MR creation, through internal discussions, self-review, pair work, CI checks, or informal coordination. Industrial teams may also rely more heavily on pre-validation practices, which reduce the amount of change visible after submission. Therefore, the absence of post-submission changes should not automatically be interpreted as the absence of review effort.

At the same time, low post-submission change rates can have different meanings. They may indicate that MRs are well prepared before review and can be accepted with little additional work. They may also indicate that reviews are lighter and that some issues are not corrected before integration. The amount-of-change measure cannot distinguish between these explanations on its own, because they differ mainly in what happens before MR submission or after merge. This observation motivates the broader process-oriented perspective developed in the remaining chapters, especially the analyses that relate peer review decisions to MR types, workflow deviations, upstream planning, and downstream CI/CD outcomes.

Overall, the amount of post-submission change is not a perfect proxy for peer review effort, and it should not be used as a standalone measure of review cost. Its contribution is more specific: it captures a distinct, frequent, and early-identifiable dimension of review-induced work. The empirical results show that this dimension is different from time to complete peer

review and from participation-based indicators, and that it can be predicted at MR creation time with an AUC of up to 0.88. These findings support the use of change-based effort as one complementary view of peer review effort, to be interpreted together with temporal measures, MR-type stratification, deviation-aware filtering, and upstream and downstream process signals.

5.6 Implications

5.6.1 Implications for our industrial partners

Although this chapter uses open-source GitLab projects for its empirical analysis, the findings carry practical implications for our industrial partners. First, the independence between post-submission change and time to complete peer review suggests that industrial teams should not rely on review duration alone when monitoring peer review effort; complementing time-based dashboards with a change-based indicator provides a more complete view of the work that reviews actually generate. Second, the predictive results show that MRs likely to require substantial post-submission change can be identified at creation time with good accuracy, which enables earlier planning decisions such as prioritizing reviewer attention toward high-change candidates, alerting authors to potential rework needs, or scheduling additional review capacity for changes that are predicted to evolve significantly during review. Finally, the feature analysis highlights complexity, author experience, and textual characteristics of the submission as the most consistent drivers of high-change MRs, which can help industrial partners focus their process-improvement efforts on the factors with the strongest practical impact on review-related rework.

5.6.2 Implications for researchers

For researchers, the main implication of this chapter is methodological: peer review effort should not be approximated by a single temporal indicator. The amount of changes required to complete peer review captures a dimension of review-generated work that is largely independent of time to complete peer review and of the number of reviewers, which means that these measures

are not interchangeable proxies of the same underlying construct. Future empirical studies on peer review effort should therefore either report both time-based and change-based indicators explicitly, or motivate their choice in terms of the specific aspect of review effort they aim to analyze. Treating change-based and time-based effort as complementary rather than substitute constructs can reduce the risk of inconsistent findings across studies that rely on different operationalizations of the same concept.

The predictive results also carry implications for the design of future peer review studies. The fact that MRs likely to require substantial post-submission change can be identified at creation time with an AUC of up to 0.88 suggests that creation-time features provide a usable signal for early-intervention studies, such as reviewer recommendation, effort-aware prioritization, or controlled experiments on review workload allocation. The consistency of complexity, author experience, and textual features as dominant drivers across contexts provides a candidate baseline feature set that other empirical studies can adopt and compare against, supporting replication and cross-project comparison. Finally, because the observed rates of high-change MRs differ substantially between open-source and industrial settings, researchers should avoid generalizing effort-related findings across contexts without explicitly reporting these base rates and, where possible, stratifying their analyses accordingly.

5.7 Threats to Validity

Internal validity threats: These may stem from implementation errors and model choices. We reduced such risks by using established metric definitions, widely adopted classifiers, 100-bootstrapping for reliability, and well-known Python packages (e.g., Scikit-learn). Nonetheless, unnoticed errors and context-specific result variations remain possible.

External validity threats: These concern the generalizability of our findings. peer review metrics based on the amount of changes can vary across project types and team structures. To mitigate this, we analyzed over 23.6K reviews across multiple projects and dimensions. However, expanding the feature set or studying other platforms (e.g., Gerrit) could improve coverage.

5.8 Conclusion

In this chapter, we examined peer review effort through a time-independent perspective based on the amount of changes required to complete peer review. The objective was to move beyond temporal approximations of effort and to study a measure that more directly reflects the corrective and adjustment work performed before an MR reaches acceptance. From a software process improvement perspective, such a measure is important because it captures a dimension of review effort that may remain invisible when effort is inferred only through time to complete peer review.

Having established a time-independent view of effort, the thesis now turns to an important source of variation in that effort: the type of MR being reviewed. The next chapter therefore examines how peer review effort and review behavior differ across categories of MRs, with particular attention to development, configuration, and documentation changes.

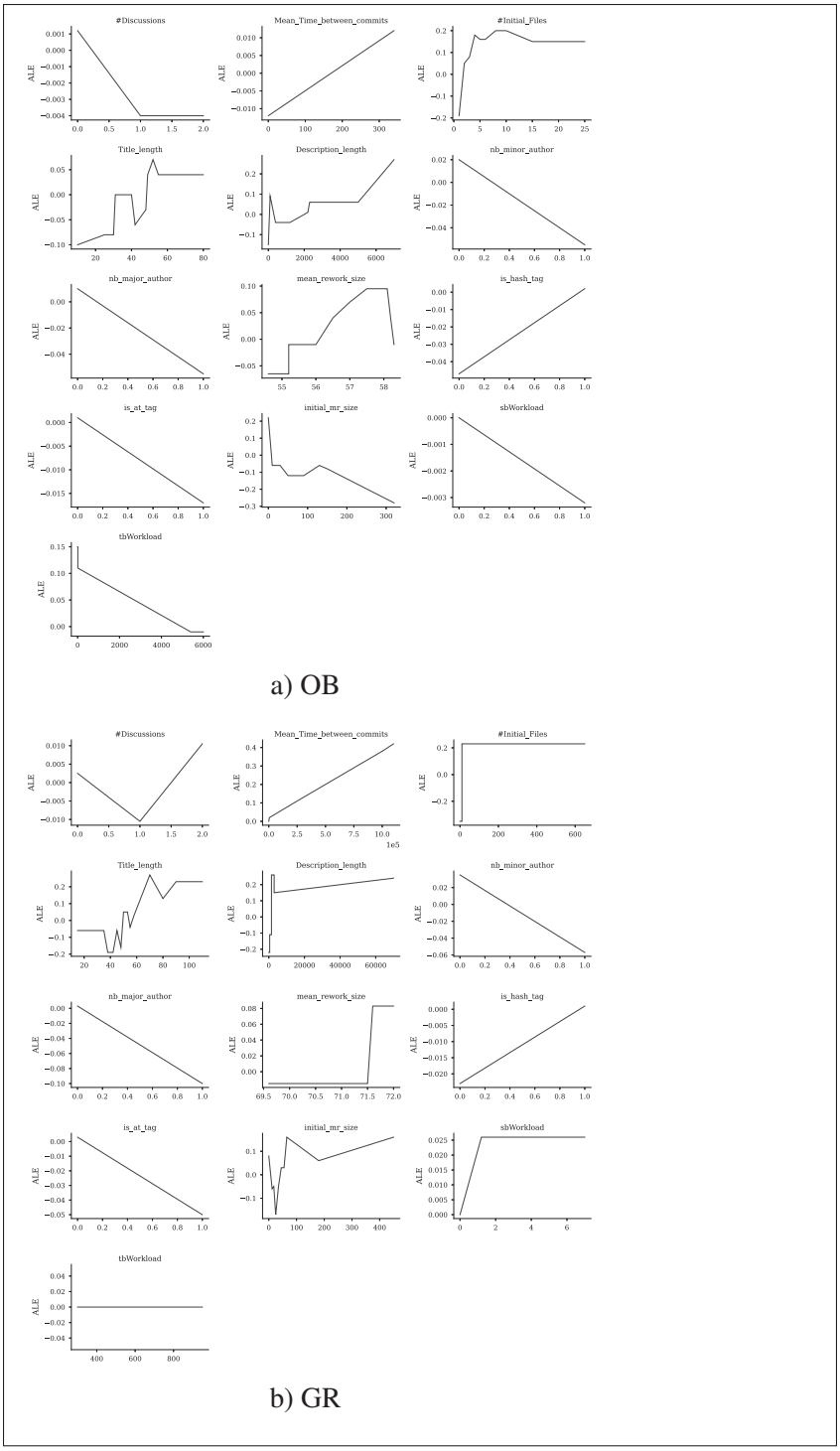
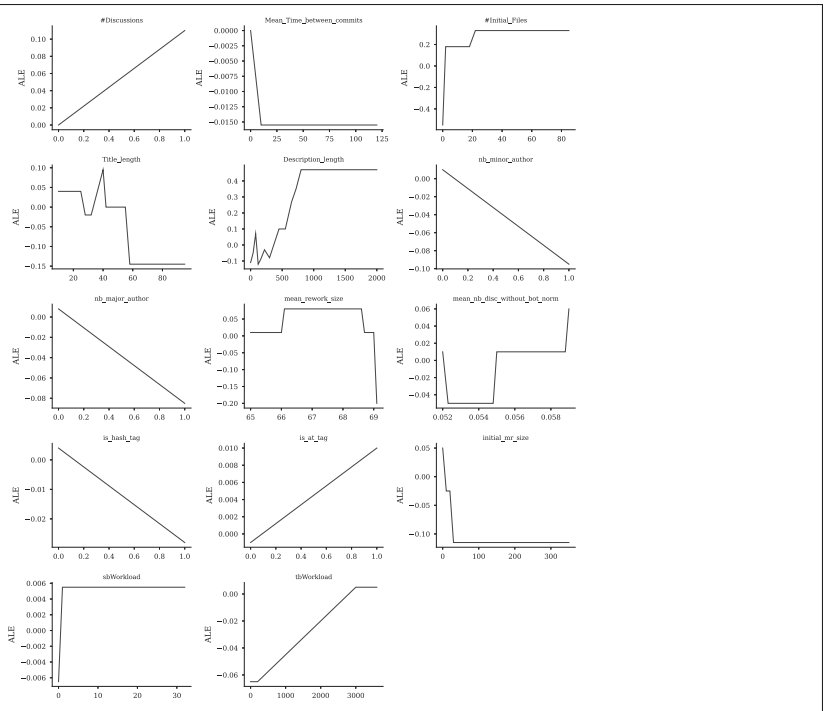
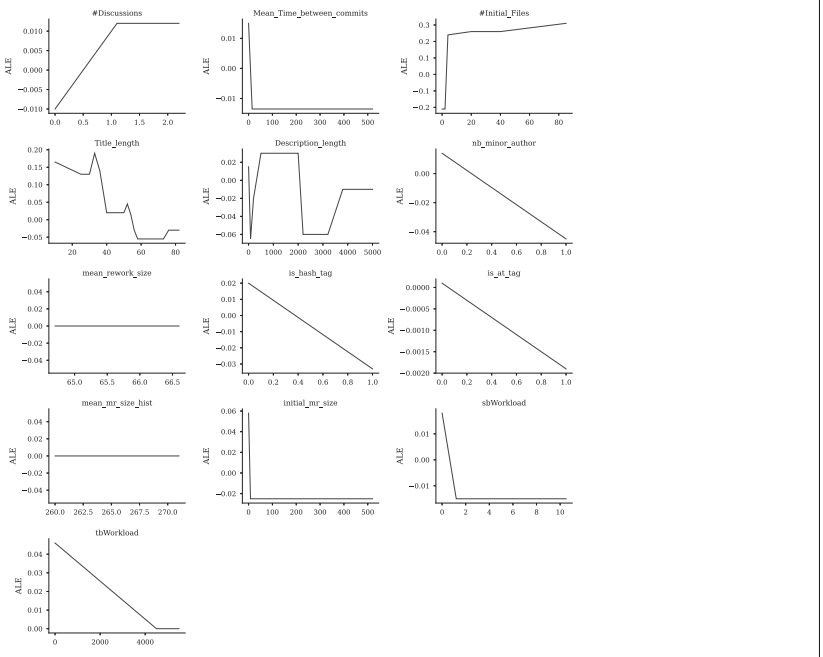


Figure 5.8 Feature importance impact on the classification model (1/2)



c) Ink



d) GT

Figure 5.8 Feature importance impact on the classification model (2/2)

CHAPTER 6

DO SDN CONFIGURATION CHANGES GET REVIEWED DIFFERENTLY? AN EMPIRICAL STUDY AT TELUS

Samah Kansab[1] , Henri Aïdasso[1] , Francis Bordeleau[1] , Ali Tizghadam[2]

[1] Department of Software Engineering and IT, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[2] TELUS, 25 York St, Toronto, Ontario, Canada M5J 2V5

Paper Published in the Empirical Software Engineering Journal on June 2026

Preamble

Peer review effort is not uniform across all MRs. This chapter examines how review activity, engagement, timeline, and quality differ across development, configuration, and documentation MRs at our industrial partner TELUS, and how bug presence modulates these patterns. Configuration-dominant MRs receive less review attention than development MRs, while configuration-inclusive MRs attract more; both configuration types involve more self-management. Bug presence amplifies review intensity and quality for configuration MRs. Replicating the analysis on open-source projects confirms the main patterns beyond the industrial context.

This chapter is adapted from a paper under second-round review at EMSE. The thesis version revises the introduction, conclusion, and overall framing to align the chapter with the objectives of this thesis.

6.1 Introduction

The previous chapter showed that peer review effort cannot be adequately understood through time to complete peer review alone. By introducing the amount of post-submission change as a time-independent indicator, it established that peer review effort includes a corrective-work dimension that remains partly invisible in temporal measures. However, measuring effort more directly raises a further question: is this effort distributed uniformly across all MRs, or does it vary according to the type of change being reviewed? Answering this question is essential for

software process improvement, because treating all MRs as a homogeneous population may hide meaningful differences in review needs, reviewer engagement, and process outcomes.

This issue is particularly important in modern DevOps environments, where software evolution depends not only on source code, but also on configuration artifacts that control building, deployment, orchestration, and runtime behavior. CI/CD pipelines, infrastructure-as-code practices, and container orchestration platforms have increased the operational importance of configuration files. In systems relying on tools such as Helm, Kubernetes manifests, deployment descriptors, or pipeline definitions, critical behavior is often encoded in configuration rather than in traditional source code alone. As a result, configuration MRs may have significant operational impact even when they appear small or syntactically simple.

This challenge is especially salient in our industrial context. Our industrial partners operate in environments shaped by Software-Defined Networking (SDN), where system behavior depends heavily on the correctness and evolution of configuration. SDN decouples the control and data planes to enable programmable network management, but this flexibility also makes configurations central to traffic flow, security policies, service interactions, and deployment conditions (Kreutz *et al.*, 2014; Liu, Kibalya, Santhosh Kumar & Zhang, 2021). In such settings, configuration cannot be treated as a marginal subtype of development work. It represents a distinct category of change whose review may require specific expertise, attention, and process support.

At the same time, configuration changes may not receive review attention proportional to their operational importance. Compared with development changes, configuration files are often shorter, less structured, less supported by dedicated testing infrastructure, and sometimes perceived as secondary to source code. In SDN systems, reviewers may also need to reason about both networking concepts and configuration-specific semantics. These characteristics suggest that configuration MRs may exhibit review patterns that differ from development or documentation MRs, and that analyzing all MRs together may obscure important differences.

Motivated by this observation, this chapter examines peer review effort and review quality across three major categories of MRs: development, configuration, and documentation. This comparison is necessary to determine whether different types of changes require different levels of reviewer attention, review strategies, or process support. In the progression of the thesis, this chapter therefore moves from measuring peer review effort more directly to identifying where this effort is concentrated, where it may be insufficient, and whether a uniform review process is appropriate across all MR types.

Although configuration review is especially important in our SDN-oriented industrial context, its relevance extends beyond this domain. The broader adoption of deployment descriptors, service configuration, infrastructure templates, and CI/CD artifacts has made configuration changes increasingly visible across many software ecosystems. For this reason, this chapter also extends the analysis to open-source GitLab projects. This extension helps assess whether the review patterns observed in the industrial context are specific to SDN-centered environments or reflect a broader phenomenon in pipeline- and infrastructure-intensive software development.

This chapter empirically investigates these questions using MRs from TELUS, a major Canadian telecommunications company that uses GitLab for collaborative development and peer review. MRs are classified according to the proportion and nature of changed files into four groups: *configuration-dominant* MRs, where configuration files represent more than 50% of changed files; *configuration-inclusive* MRs, where configuration files are present but not dominant; *development* MRs; and *documentation* MRs. Within the configuration-related groups, we further distinguish bug-related from non-bug-related MRs to examine whether corrective changes receive different review attention.

To capture the multidimensional nature of peer review, the chapter adopts a mixed-methods design combining quantitative and qualitative analyses. Quantitatively, we evaluate twelve metrics across four dimensions: activity, engagement, timeline, and approval. Qualitatively, we assess review comment quality using a four-level rating scale and analyze comments according to four cognitive attributes: *Emotion*, *Suggestion*, *Evaluation*, and *Question* (Yang *et al.*, 2023).

This combined perspective makes it possible to examine not only how much review activity occurs, but also how reviewers engage with changes and what kind of feedback they provide.

The findings show that configuration-dominant MRs receive significantly less review activity and participation than development MRs, while configuration-inclusive MRs display higher engagement and are reviewed more intensively than both development and documentation MRs. Across both configuration categories, reviewers also engage more frequently in self-approval, self-assignment, and self-integration than for development MRs, suggesting more ownership-driven review practices. Bug presence further modulates this profile: bug-related configuration MRs attract more reviewers, receive more comments, take longer to complete peer review, and exhibit higher-quality feedback. The open-source analysis confirms the distinct profile of configuration-inclusive MRs, while also showing context-specific differences in how configuration-dominant changes are reviewed.

The main contributions of this chapter are:

- a taxonomy for categorizing MRs into development, documentation, configuration-dominant, and configuration-inclusive changes;
- empirical evidence that configuration MRs occupy a distinct review profile, differing from both development and documentation MRs in review activity, engagement, timeline, and self-management practices;
- evidence that bug presence modulates both the intensity and quality of configuration review, showing that reviewer attention is not distributed uniformly across configuration work;
- an extension of the analysis to open-source GitLab projects, supported by a replication toolkit, showing that configuration review is not limited to a single industrial context but reflects a broader software engineering concern.

6.2 Study Design

In this chapter, we investigate whether peer review effort and review quality vary according to the type of MR under review. Building on the previous chapter, which established a time-

independent perspective on peer review effort, the objective here is to determine whether that effort is distributed uniformly across MR categories or whether different types of changes exhibit distinct review patterns. To address this question, we design a mixed-methods study combining quantitative and qualitative analyses of merge requests.

The study relies on MR and issue data collected from TELUS, as described in Section 3.3.2. To support the comparison across merge request categories, we first enrich MRs with issue information and bug-related labels, then classify them into configuration, development, and documentation categories. We thereafter analyze the peer review process through two complementary lenses. The quantitative analysis compares review activity, engagement, timeline, and approval-related metrics across MR categories. The qualitative analysis examines the quality and cognitive characteristics of review comments in a statistically representative sample.

6.2.1 Research Questions

The chapter is guided by the following four research questions:

RQ1 **What are the key quantitative differences in the peer review process between configuration MRs and those related to development or documentation?** This question examines whether MR categories differ in terms of review activity, engagement, timeline, and approval behavior. Since the previous chapter showed that peer review effort should not be reduced to a single temporal measure, it becomes important to determine whether different categories of changes also exhibit different process profiles. Comparing configuration, development, and documentation MRs helps identify whether a uniform review process is appropriate across all change types or whether more differentiated review practices are needed.

RQ2 **How does the quality of peer review vary between configuration MRs and those related to development and documentation?** Quantitative differences alone do not fully characterize the review process. Two MR categories may receive comparable amounts of

activity while still differing in the depth, usefulness, or cognitive nature of the feedback they attract. This question therefore examines whether configuration-related MRs receive review comments of similar quality to those observed in development and documentation reviews.

RQ3 What are the differences in the peer review process between bug-related and non-bug-related configuration MRs? Bug-related changes often represent more urgent or more operationally sensitive work than routine updates. Investigating whether bug presence changes reviewer attention, engagement, or time to complete peer review is important for understanding whether configuration review effort is shaped not only by artifact type, but also by the purpose of the change.

RQ4 How does the quality of peer review differ between bug-related and non-bug-related configuration MRs? If bug-related configuration MRs receive more reviewer attention, it is also important to determine whether this attention translates into qualitatively different feedback. This question therefore examines whether bug presence is associated with deeper, more technical, or otherwise different forms of peer review feedback.

6.2.2 Data Mapping and Enrichment

To support the analyses in this chapter, we first enrich the MR data with issue associations and bug-related labels.

6.2.2.1 Associating MRs and Issues

Each MR is expected to be associated with a relevant issue that describes the underlying task, bug, or feature request. To determine these associations, we follow a bi-directional parsing strategy using MR and issue metadata.

Forward Matching (MR to Issue). We examine each MR's title, description, and commit messages for references to issue identifiers. These references may appear in two common formats:

- Short reference: `#IssueID` (e.g., `closes #1` indicating closure of issue 1)
- Full URL reference: `https://<gitlab>/issues/#IssueID`

When a reference is detected, we match it against the issue dataset and extract the corresponding issue metadata.

Reverse Matching (Issue to MR). To improve recall, we also parse issue titles and descriptions and search for references to specific MRs (e.g., MR links or MR IDs). This additional step helps uncover associations that may have been initiated from the issue side, such as follow-up development MRs mentioned by stakeholders.

Label Enrichment. Once MR/issue associations are determined, we enrich the MR metadata with the labels (e.g., `bug`, `enhancement`, `performance`) attached to the associated issues. These labels provide essential context for downstream classification and analysis, including bug identification and MR type categorization.

6.2.2.2 Identifying MRs with Bug

To confidently identify bug-related MRs, we employ a multi-stage classification strategy combining label-based matching, linked issue analysis, and a keyword-based heuristic validated through manual inspection.

Label-Based Classification. If an MR is explicitly labelled as `bug`, it is directly classified as bug-related. Similarly, if the MR is linked to an issue labelled as `bug`, we inherit this label and classify the MR as bug-related. As a result, this classification is valid for 80% of the MRs.

Heuristic-Based Classification. Approximately 20% of MRs in our dataset are neither labelled nor linked to any issue. To classify these unlabelled MRs, we apply a keyword-based heuristic inspired by prior research in defect and bug-fix identification Wang *et al.* (2021); Tian,

Lawall & Lo (2012); Kim, Zimmermann, Whitehead Jr & Zeller (2007); Lukins, Kraft & Etzkorn (2008); Marcus, Sergeyev, Rajlich & Maletic (2004); Jiang *et al.* (2025); Tufano, Pantiuchina, Watson, Bavota & Poshyvanyk (2019a); Avula & Mondal (2024). We compiled and refined a list of commonly used bug-related terms from these studies and further validated them with our industrial collaborators.

The following keywords are used to identify bug-related MRs:

- **Core terms:** `fix`, `bug`, `resolved`, `solve`, `debug`, `bugfix`, `problem`, `error`, `defect`, `mistake`, `correct`, `fault`
- **Expanded terms:** `broken`, `failure`, `regression`, `patch`, `hotfix`, `workaround`

We consider an unlabelled MR to be bug-related if any of these keywords appear in at least one of the following MR's fields: `title`, `description`, if not, we alos look at the associated `commit messages`. This heuristic enables us to identify remaining likely bug-fixing MRs that lack formal annotations.

Manual Validation. To validate the accuracy of this approach, we randomly sampled 50 previously unlabelled MRs that were classified as bug-related by our heuristic. Manual inspection by two researchers, with disagreements resolved through discussion, confirmed that **46/50 (92%)** were correctly classified. Given $n = 50$, this estimate should be interpreted as a sanity check rather than a precise measurement: under a binomial model, the corresponding 95% confidence interval is approximately 92% ± 8% (i.e., $\sim [84\%, 100\%]$), and the worst-case margin of error at $n = 50$ is about ±14% when the true proportion is near 0.5. Overall, this result provides additional confidence that our keyword-based heuristic reliably captures bug-fix intent when explicit labels or issue links are unavailable.To also validate the inverse case, we applied the same procedure to 50 previously unlabelled MRs classified as non-bug-related and found that **47/50 (94%)** were indeed not bug-fix related (95% CI $\approx$ 94% ± 7%). Overall, these checks increase confidence that our keyword-based heuristic captures bug-fix intent when explicit MR labels or reliable issue links are unavailable.

The complete classification workflow is illustrated in Figure 6.1.

Table 6.1 Overview of the number of MRs per category

Category	Config		Non-Config	
	config-d	config-i	dev	doc
Total MRs	2,613 (49%)	2,699 (51%)	2,797 (87%)	386 (13%)
Bug-related MRs	695 (26%)	1,341 (49%)	–	–
Overall	5,312 (62%)		3,183 (38%)	

config-d: configuration-dominant; config-i: configuration-inclusive; dev: development; doc: documentation.

Table 6.2 Summary of most frequent file extensions, configuration file types, and DevOps/configuration tools per repository group

Repository Group	Top Extensions Touched	Top Config File Types	Tools Detected
NetApps	`py` (1635), `tsx` (556), `ts` (465), `json` (240), `yaml` (230)	`json` (240), `yaml` (230), `xml` (216), `yml` (46), `dockerfile` (36)	`Docker`, `docker compose`, `GitLab CI`, `Helm`, `OpenShift`
Platform	`py` (834), `yaml` (667), `yml` (220), `sh` (152), `json` (123)	`yaml` (667), `yml` (220), `json` (123), `dockerfile` (40), `tpl` (37)	`Docker`, `docker compose`, `GitLab CI`, `Helm`, `OpenShift`
Infrastructure	`tf` (406), `yaml` (165), `tfvars` (102), `dockerfile` (24), `py` (9)	`tf` (406), `yaml` (165), `tfvars` (102), `dockerfile` (24), `json` (7)	`Docker`, `docker compose`, `GitLab CI`, `Helm`, `Kustomize`, `OpenShift`, `Terraform`
CDAF	`xml` (644), `py` (402), `yaml` (174), `tsx` (163), `java` (105)	`xml` (644), `yaml` (174), `yml` (94), `json` (66), `properties` (25)	`Docker`, `docker compose`, `GitHub Actions`, `GitLab CI`, `Helm`, `OpenShift`, `Terraform`
CNE-IP	`noext` (127), `py` (126), `json` (75), `tsx` (52), `pyc` (49)	`json` (75), `yaml` (19), `yml` (15), `cfg` (5), `dockerfile` (2)	`Ansible`, `Docker`, `docker compose`, `GitLab CI`, `Helm`, `OpenShift`

6.2.3 MR Categorization

To refine our analysis, we filtered the dataset to focus on specific types of MRs, as summarized in Table 6.1. Our goal is to distinguish between configuration, development, and documentation-related MRs, as well as configuration-related MRs that include other types of changes. The classification criteria were defined in collaboration with domain experts at our industrial partner and are described below.

6.2.3.1 Identifying Configuration MRs

We used three heuristics to identify MRs that modify configuration files:

- **File Extensions:** Files with configuration-specific extensions, including `cfg`, `toml`, `ini`, `json`, `yml`, `yaml`, `eslintrc`, `prettierrc`, `dockerfile`, `npmrc`, `gitattributes`, `dockerignore`, `helmignore`, `flake8`, `srl`, and `gotmpl`, are identified based on internal conventions.
- **File Naming:** Files with common code extensions (e.g., `.py`) but named exactly `config.[ext]` (e.g., `config.py`) are considered configuration files.
- **Folder Naming:** Files located within directories explicitly named `config/` (without suffixes or prefixes) were also classified as configuration files (e.g., `project/config/file.py`).

If at least one configuration file is present, the MR is considered configuration-related. For each configuration-related MR, we calculated the ratio of changed configuration files to all changed files. Then, we further distinguished:

- **Configuration-dominant (config-d) MRs:** A ratio ≥ 50% of changed files are configuration files
- **Configuration-inclusive (config-i) MRs:** A ratio < 50% of changed files are configuration files

Our classification method relies on the dominant file type ratio to ensure consistency across projects. We define configuration-dominant MRs using a ≥50% threshold because it provides an intuitive and interpretable boundary: configuration files constitute the majority of the modified artifacts in the MR, which aligns with our objective of separating MRs where configuration is likely the primary focus from those where it is mixed with other changes. To ensure that this operational choice does not drive the conclusions, we performed a sensitivity analysis by repeating the main comparisons with alternative thresholds (40%, 45%, 55%, and 60%), and we observed the same overall trends; these robustness results are now reported in the revised manuscript.

While churn-based classification (based on the number of lines changed) is a valid alternative, it introduces interpretability challenges. For example, a single-line configuration change may carry significant impact and demand extensive review, especially in critical projects. An MR

may include only two lines of configuration changes and ten lines of documentation changes, yet the review discussion may focus almost entirely on the configuration aspect. To explore this dimension, we analyzed configuration churn as a complementary signal. We found that 35% of configuration-related MRs involved more than 50% configuration churn, and that 51.44% of configuration-related MRs modified at least 40% of their content as configuration churn, highlighting the frequent involvement of configuration artifacts in SDN changes. At the same time, configuration file edits are often incremental in terms of lines: 49.02% of MRs involve initial configuration changes below 10% of total churn, and 89.8% exhibit configuration rework below 10%. This partial alignment between churn-based and file-ratio-based views provides supporting evidence for our chosen approach while also illustrating why churn alone can understate the importance of configuration changes. Nonetheless, we acknowledge the limitations of any single proxy and plan to explore more comprehensive churn- and semantics-aware classification strategies in future work.

6.2.3.2 Identifying Documentation MRs

Documentation MRs are identified using file extensions commonly used for textual and visual documentation. These include:

- **Textual formats:** `md`, `mdx`, `rst`, `mjml`, `txt`, `docx`
- **Media formats:** `png`, `jpg`, `ico`, `svg`, `gif`

An MR is classified as documentation-related if over 50% of its modified files are in standard documentation formats (e.g., .md, .rst, .txt). This classification excludes inline documentation embedded within source code. While in our context, such comments are common, they primarily serve as localized annotations and do not typically represent formal documentation practices. Furthermore, the projects under study systematically separate documentation into standalone files.

6.2.3.3 Identifying Development MRs

Any MR that do not meet the criteria for configuration or documentation is classified as development-related. These typically involve changes to application logic, test code, or infrastructure scripts. We confirmed with project maintainers that development work is generally done in file types not captured by the configuration or documentation filters. MRs with less than 50% documentation files, and no configuration files, fall into this category.

This classification system ensures that we capture meaningful differences in review behavior across distinct artifact types, supporting a fair and relevant comparative analysis of review activity, engagement, and quality.

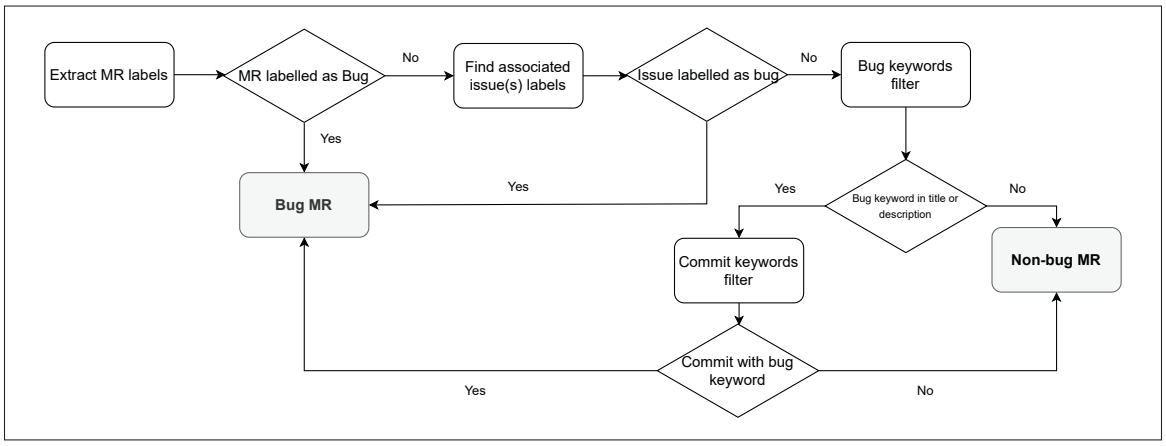


Figure 6.1 Bug identification flow diagram

6.2.3.3.1 Classification validation and rationale

Because our MR categorisation is automated, we added an explicit validation step to ensure the labels are consistent with the underlying file changes. Concretely, we implemented a set of Python verification tests (invariant checks, similar in spirit to unit tests) that re-scan each MR *after* classification and confirm that the assigned label matches the observed diff content. For example, the tests verify that every MR labelled as configuration-related modifies at least one file matched by our configuration-file heuristics, and that the computed configuration-file ratio (used to separate config-d and config-i) is consistent with the category assignment. To further

contextualize the artifacts involved, we also report the most frequent file extensions observed in the studied projects in Table 6.2.

6.2.4 Analysis Strategy

6.2.4.1 Quantitative Analysis

Metrics Extraction. We extracted 15 metrics essential for comparing the review process across different types of MRs, reported in Table 6.3. For example, the time to complete peer review—defined as the duration between MR creation and closure (either merged or closed)—is not directly available from the GitLab API and must be computed manually. The extracted metrics are categorized into four key dimensions as follows:

- **Activity:** Captures the volume and nature of review activity, such as the number of comments and the extent of rework.
- **Engagement:** Measures the degree of participation in the review process, including the number of committers, reviewers, and collaborators.
- **Timeline:** Reflects the duration of the review process, including the time to first review and time to completion.
- **Approval:** Assesses the review outcome, including whether the MR was approved, and whether it was self-managed (i.e., self-approved, self-merged, or self-assigned).

Metrics Analysis. To examine differences across MR types—such as configuration versus development or documentation, and bug-related versus non-bug-related—we apply statistical tests to compare the extracted metrics.

First, we assess the distribution of each metric using the Shapiro-Wilk test Shapiro, Wilk & Chen (1968), which is widely regarded as one of the most powerful and reliable tests for normality in small to moderate-sized samples Razali, Wah *et al.* (2011); Kitchenham *et al.* (2017). Our results confirmed that the majority of metrics are not normally distributed ($\alpha = 0.05$).

Given the non-parametric nature of the data, we choose the Mann-Whitney U test as our primary statistical test. The Mann-Whitney U test is especially appropriate for comparing independent samples where the goal is to determine whether one distribution tends to have larger or smaller values than another.

Following the approach of (Thongtanunam *et al.*, 2015), we use a one-tailed Mann–Whitney U test at a significance level of $\alpha = 0.05$ to determine whether differences between groups are statistically significant and, if so, in which direction (e.g., whether configuration MRs tend to receive more or less review activity than development MRs). To complement statistical significance with practical relevance, we report an effect size using the *rank-biserial correlation* (r_{rb}) Radliński (2024); Fister (2007), derived from the U statistic, where the sign indicates the direction (positive values mean higher values in the first group) and the magnitude indicates the strength of the difference. We interpret $|r_{\text{rb}}|$ using standard thresholds: *negligible* (< 0.147), *small* ($[0.147, 0.33)$), *medium* ($[0.33, 0.474)$), and **large** (≥ 0.474).

6.2.4.2 Qualitative Analysis

Review Collection. To assess the quality of review comments (or simply review in the following), we collect all discussion comments associated with each MR along with the list of participating reviewers' metadata. We filter out comments authored by the MR's author to focus exclusively on feedback provided by external reviewers. This ensures that we analyze only MRs that undergo an actual peer review process, rather than internal notes or self-commentary.

We further refine the dataset by excluding automated system-generated messages, which are flagged by the GitLab API using the attribute system=true. Examples include boilerplate notifications such as *"approved this MR"* or *"enabled an automatic merge when the pipeline for..."*. This step ensures that the dataset includes only substantive, manually written reviewer comments relevant to our analysis.

Review Sample Selection. Manually analyzing all reviews is impractical. So, following prior work Thongtanunam *et al.* (2015); Beller, Bacchelli, Zaidman & Juergens (2014), we follow a

Table 6.3 Review-process metrics grouped by analysis dimensions (D)

D	Metric (definition)	Rationale
Activity	**#Comments:** total discussion comments on the MR.	Proxy for discussion intensity and review effort.
	#Reviewer comments: comments authored by human reviewers (excluding bots).	Captures peer feedback directly tied to review quality.
	MR size: initial churn (additions + deletions) before review iterations.	Baseline change magnitude; often relates to review load.
	#Additions: lines added before review iterations.	Measures introduced content (implementation / integration surface).
	#Deletions: lines deleted before review iterations.	Measures removed/replaced content (refactoring or corrective removal).
	Rework size: churn introduced after MR creation during review iterations.	Captures revisions needed to satisfy review feedback.
Engage.	**#Committers:** distinct contributors who pushed commits to the MR.	Indicates coordination breadth and shared ownership.
	#Reviewers: distinct reviewers participating in the review (e.g., commenting/approving).	More reviewers can increase scrutiny and defect detection.
	#Collaborators: distinct non-author participants contributing to the discussion.	Reflects cross-role involvement beyond the core review.
Timeline	**Time to complete:** hours from MR creation to merge/close.	Measures end-to-end review turnaround and potential bottlenecks.
	Time to first review: hours from creation to the first review comment.	Early feedback is typically associated with faster resolution.
Approval	**Approved:** MR received at least one formal approval (binary).	Captures compliance with approval gates.
	Self-approved: author approved their own MR (binary).	May weaken independent peer validation.
	Self-merged: author merged their own MR (binary).	Indicates reduced separation of duties / process exceptions.
	Self-assigned: author assigned the MR to themselves (binary).	Proxy for autonomy and reviewer availability constraints.

statistically grounded sampling strategy to select a representative subset of reviews for manual annotation.

We compute the required sample size using the standard formula for estimating a proportion within a finite population:

$$s = \frac{z^2 \cdot p \cdot (1 - p)}{c^2} \tag{6.1}$$

where z is the Z-score corresponding to the desired confidence level (1.96 for 95%), p is the estimated population proportion (set to 0.5 for maximum variability), and c is the margin of error (set to 0.05).

Because the review population is finite, we apply the finite population correction as follows:

$$ss = \frac{s \cdot P}{P + s - 1} \tag{6.2}$$

where P is the total number of reviews. We round the result to the nearest whole number, resulting in a final sample size of 375 reviews.

Annotation Process and Validation. The first two authors manually annotated the 375 sampled reviews, representing the 14,206 reviews. Each review is independently assessed according to predefined criteria for review quality and content attributes, detailed in the next sections. To ensure consistency and domain relevance, we validate the annotations through iterative discussions with industrial collaborators.

We measure inter-rater reliability using Cohen's kappa coefficient, which yields a value of 0.95—indicating almost perfect agreement. This result supports the reliability of the manual annotation and the validity of the qualitative analysis.

Review Quality Analysis. To analyze the quality of reviews, we characterize and assign a grade to each review. We consider four grades for the review quality:

- **Poor:** This grade represents the lowest quality of review. Comments in this category are often unhelpful, lacking detail, unprofessional, or irrelevant to the changes being reviewed.
- **Fair:** This grade indicates a review of moderate quality. The comments may be somewhat useful but still lack depth or specificity. The review is helpful but does not provide a comprehensive evaluation of the changes.
- **Good:** This grade reflects a solid, reliable review. Comments are generally detailed, accurate, and relevant, providing constructive feedback that the author can use to improve the code.
- **Excellent:** This grade represents the highest quality of review. Comments are thorough, insightful, and demonstrate a strong understanding of the code and its context. The feedback is highly relevant, well-structured, and offers valuable suggestions for improvement.

For review comment characterization, we consider the same four attributes used by (Yang *et al.*, 2023), who apply the data triangulation method Thurmond (2001) based on authoritative documents, academic literature, and real-world examples to effectively define the following attributes that characterize reviews:

- **Emotion:** Reviews should evaluate the code itself rather than address the author personally. For example, instead of saying, "You missed adding feature Y in function X," a more appropriate comment would be, "Function X is missing feature Y."
- **Question:** Reviewers can ask questions to clarify unclear parts of the code or to gain new knowledge. For example, a reviewer might ask, "Can you explain why you chose this approach for handling error X?" to understand the rationale and ensure the robustness of the solution, or to gain more knowledge on the used method.
- **Evaluation:** Reviews should assess the code's weaknesses and identify areas for validation, improvement, or removal. For instance, a reviewer might note, "The current implementation of function Y introduces a performance bottleneck. Consider optimizing it or using an alternative method."
- **Suggestion:** Reviewers are expected to offer suggestions for solving problems or making improvements during the review process. For example, "Consider refactoring this section of the code to improve readability and maintainability by splitting it into smaller functions."

The reviews are blindly evaluated (i.e., without seeing the types of MRs) during the manual annotation. Then, we validate the review categorization with our industrial collaborators through regular meetings and discussions, ensuring that the manual annotations align with their practices and expectations, to improve the practical significance of our analysis.

6.3 Results

6.3.1 Quantitative analysis of configuration vs. non-configuration MRs

RQ1: What are the key quantitative differences in the review process between configuration MRs and those related to development and documentation

To answer RQ1, we compare review-process and review-activity metrics across MR types (configuration-dominant, configuration-inclusive, development, and documentation), as defined in Section 6.2.3. We report descriptive statistics and assess between-group differences using non-parametric tests (Mann–Whitney U for pairwise comparisons) together with effect sizes, since the metric distributions are non-normal. The complete list of metrics is provided in Table 6.3, and the statistical procedures are detailed in Section 6.2.4.1.

6.3.1.1 Configuration vs Development

Configuration-dominant MRs exhibit significantly lower review activity than development MRs (p-value<0.001), while configuration-inclusive MRs show significantly higher activity. Table 6.4 indicates that configuration-dominant MRs are significantly smaller than development MRs (lower MR size and additions) and tend to include relatively more deletions, suggesting that they more often *adjust, remove, or rectify* existing settings rather than introduce substantial new functionality Kansab, Sayagh, Bordeleau & Tizghadam (2025d); Staron *et al.* (2013). Consistent with this change profile, configuration-dominant MRs receive fewer comments (overall and from reviewers) and require less rework (in lines of code) than development MRs. This aligns with the nature of configuration-dominant changes, which are frequently narrow and parameter-centric (e.g., updating a value such as `db_port = 5432`), and can be validated by a small set of specialists with limited need for extended discussion or iterative revisions.

In contrast, configuration-inclusive MRs typically combine configuration updates with surrounding implementation and validation work (e.g., code adaptations, tests, and integration changes required to make the configuration effective). Such MRs therefore affect a broader set of artifacts

Table 6.4 One-tailed Mann–Whitney U test results comparing configuration and development MRs

Dimension	Metric	config-d vs. dev	config-i vs. dev
Activity	#Comments	config-d < dev ***	*config-i > dev* ***
	#Reviewer's comments	config-d < dev ***	config-i > dev ***
	MR size	*config-d < dev* ***	**config-i > dev *** **
	#Additions	*config-d < dev* ***	**config-i > dev *** **
	#Deletions	*config-d > dev* ***	**config-i > dev *** **
	Rework size	*config-d < dev* ***	**config-i > dev *** **
Engagement	#Committers	config-d > dev ***	*config-i > dev* ***
	#Reviewers	config-d < dev ***	config-i > dev *
	#Collaborators	*config-d < dev* ***	config-i > dev ***
Timeline	Time to complete peer review	config-d < dev ***	*config-i > dev* ***
	Time to first review	config-d < dev ***	config-i > dev ***
Approval	Approved	config-d < dev ***	config-i > dev ***
	Self-Approved	*config-d > dev* ***	config-i > dev ***
	Self-Merged	*config-d > dev* ***	config-i > dev **
	Self-Assigned	*config-d > dev* ***	config-i > dev ***

config-d: configuration-dominant; config-i: configuration-inclusive; dev: development.
Statistical significance: * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$.
Effect size: normal = negligible, *italic* = small, underline = medium, **bold** = large.

and stakeholders, creating more coordination needs and more opportunities for reviewers to provide feedback anchored in executable code and tests. This broader scope plausibly explains why configuration-inclusive MRs show significantly higher churn and review activity (larger MR size, more additions/deletions, more comments, and more iterations) compared to development MRs.

Configuration-dominant MRs receive less attention from reviewers compared to development and configuration-inclusive MRs. Table 6.4 shows that both configuration-dominant and configuration-inclusive MRs involve significantly more committers than development MRs (p-value<0.001), suggesting that configuration work often requires coordinated contributions across multiple components (e.g., updating configuration alongside dependent scripts, templates,

or validation assets). However, configuration-dominant MRs attract significantly fewer reviewers and collaborators than development MRs (p-value<0.001), which is consistent with a specialization effect where configuration changes are owned and assessed by a narrower set of domain experts. Configuration-inclusive MRs attract slightly more reviewers than development MRs (p-value<0.05), likely because their mixed nature increases relevance to more stakeholders; nevertheless, reviewer engagement may still not scale proportionally with the number of committers because end-to-end review remains constrained by the need for SDN/configuration expertise.

Interestingly, configuration-dominant MRs are reviewed and receive their first comment more quickly than development MRs, as shown in Table 6.4. This faster feedback is consistent with configuration-dominant MRs involving fewer participants and minimal rework, which reduces coordination overhead and keeps the discussion scope narrow, thereby accelerating both first response and time to complete peer review. In contrast, configuration-inclusive MRs take significantly longer to receive initial feedback and to complete review, plausibly because they combine configuration with development and documentation updates; this broader scope requires aligning reviewers across multiple concerns (configuration, code, and tests) and typically triggers additional clarification and validation cycles that prolong the time to complete peer review.

Both configuration-dominant and configuration-inclusive MRs receive fewer approvals and are more self-managed compared to development MRs. Table 6.4 shows that configuration-inclusive MRs are significantly more likely to be approved than development MRs (p-value <0.001), whereas configuration-dominant MRs receive fewer approvals. At the same time, both configuration categories are significantly more self-assigned, self-approved, and self-integrated than development MRs (p-value <0.01), indicating a stronger tendency toward self-management. This pattern suggests that configuration work is often ownership-driven: authors who implement configuration changes may also hold the required operational context and permissions to progress the MR, reducing reliance on external decision-makers and potentially contributing to shorter review cycles. The higher approval likelihood for configuration-inclusive MRs may further reflect that mixed changes provide richer supporting context (e.g., associated code changes

Table 6.5 One-tailed Mann–Whitney U test results comparing configuration and documentation MRs

Dimension	Metric	config-d vs. doc	config-i vs. doc
Activity	#Comments	config-d > doc ***	**config-i > doc ***
	#Reviewer's comments	config-d > doc **	*config-i > doc* ***
	MR size	config-d < doc ***	config-i > doc ***
	#Additions	config-d < doc ***	**config-i > doc ***
	#Deletions	*config-d > doc* ***	**config-i > doc ***
	Rework size	config-d > doc ***	*config-i > doc* ***
Engagement	#Committers	config-d > doc ***	config-i > doc ***
	#Reviewers	config-d > doc ***	*config-i > doc* ***
	#Collaborators	config-d > doc *	*config-i > doc* ***
Timeline	Time to complete peer review	*config-d > doc* ***	config-i > doc ***
	Time to first review	config-d > doc **	config-i > doc ***
Approval	Approved	*config-d > doc* ***	config-i > doc ***
	Self-Approved	config-d > doc ***	*config-i > doc* ***
	Self-Merged	config-d > doc	*config-i < doc* ***
	Self-Assigned	config-d > doc ***	*config-i > doc* ***

config-d: configuration-dominant; config-i: configuration-inclusive; doc: documentation.
Statistical significance: * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$.
Effect size: normal = negligible, *italic* = small, underline = medium, **bold** = large.

and tests), making the intent and validation of the change more explicit and therefore easier to approve than configuration-only updates.

6.3.1.2 Configuration vs Documentation

Contrary to our initial findings, both configuration-dominant and configuration-inclusive MRs exhibit higher review activity and engagement than documentation MRs. Table 6.5 shows that configuration-dominant MRs are significantly smaller than documentation MRs in terms of initial MR size and additions, while exhibiting relatively higher deletions; configuration-inclusive MRs, in contrast, show significantly larger MR size, additions, and deletions, consistent with our earlier comparisons against development MRs. Despite these size differences, *both* configuration

categories receive significantly more comments, require significantly more rework, and involve significantly more committers and reviewers than documentation MRs (all p-values <0.001). While the increase in collaborators for configuration-inclusive MRs is smaller, it remains statistically significant (p-value <0.05). Overall, these results indicate that configuration-related MRs trigger richer discussion and more iterative refinement than documentation changes. This aligns with the operational role of configuration in SDN systems: even small configuration edits can alter runtime behavior, policy enforcement, and service correctness, thereby demanding careful scrutiny, clarification, and validation. Documentation updates, by comparison, are typically lower risk and easier to assess locally, which plausibly explains their lower discussion volume and rework.

Documentation MRs require less time compared to both configuration-dominant MRs and configuration-inclusive MRs. As detailed in Table 6.5, both types of configuration MRs take significantly longer to receive the first review (p-value <0.01) and to complete the review process compared to documentation MRs (p-value <0.001). This difference is consistent with the fact that documentation changes are often validated by inspection and have limited downstream operational impact on the software, whereas configuration changes in SDN settings can require additional checks and coordination (e.g., confirming policy intent and interactions or validating deployment effects), which may delay first feedback and prolong review completion even when the churn is small.

As shown in Table 6.5, configuration-dominant and configuration-inclusive MRs tend to receive more approvals but involve greater self-management compared to documentation MRs. In particular, both configuration categories are significantly more likely to be self-approved and self-assigned than documentation MRs, indicating a stronger tendency toward author-driven progression, while still receiving more approvals overall. An exception appears for self-merging: configuration-dominant MRs do not differ significantly from documentation MRs, whereas configuration-inclusive MRs are significantly less frequently self-merged. This pattern is consistent with configuration changes being operationally sensitive—thus requiring stronger gatekeeping through approvals—while also demanding specialized context that authors often

hold, leading to more self-management. The lower self-merge rate for configuration-inclusive MRs further aligns with their broader scope (mixing configuration with code and tests) and higher integration risk, which may require a more formal integration step than is typical for documentation updates.

6.3.2 Qualitative analysis of configuration vs. non-configuration MRs

RQ2: How does the quality of reviews vary between configuration MRs and those related to development and documentation

To answer RQ2, we assess review quality through a manual qualitative annotation of a stratified sample of MR review discussions (Section 6.2.4.2). Each sampled review is assigned a quality grade on a four-level scale (Poor, Fair, Good, Excellent), and we additionally code the presence of review attributes (Emotion, Question, Evaluation, Suggestion) following the adopted framework. We report grade and attribute distributions per MR type; details on sampling, annotator training, and agreement are provided in Section 6.2.4.2.

6.3.2.1 Configuration vs. Development

Figure 6.2 shows that both development and configuration-related MRs can receive high-quality reviews, but configuration-dominant MRs exhibit a noticeably less favorable distribution. Development reviews are largely rated as *Excellent* (54.5%), with additional reviews rated *Good* (9.1%), while the remaining reviews are *Fair* (31.8%) or *Poor* (4.5%). Configuration-dominant MRs present a more mixed distribution, with fewer *Excellent* reviews (34.8%) and a substantially higher share of *Fair* (26.1%) and *Poor* (15.2%) reviews. In contrast, configuration-inclusive MRs show a stronger grade distribution than configuration-dominant MRs, with 46.5% *Excellent* and 27.1% *Good* (73.6% combined), and lower *Poor* ratings (7.1%). Overall, these results suggest that the drop in perceived review quality is most pronounced when MRs are dominated by configuration artifacts.

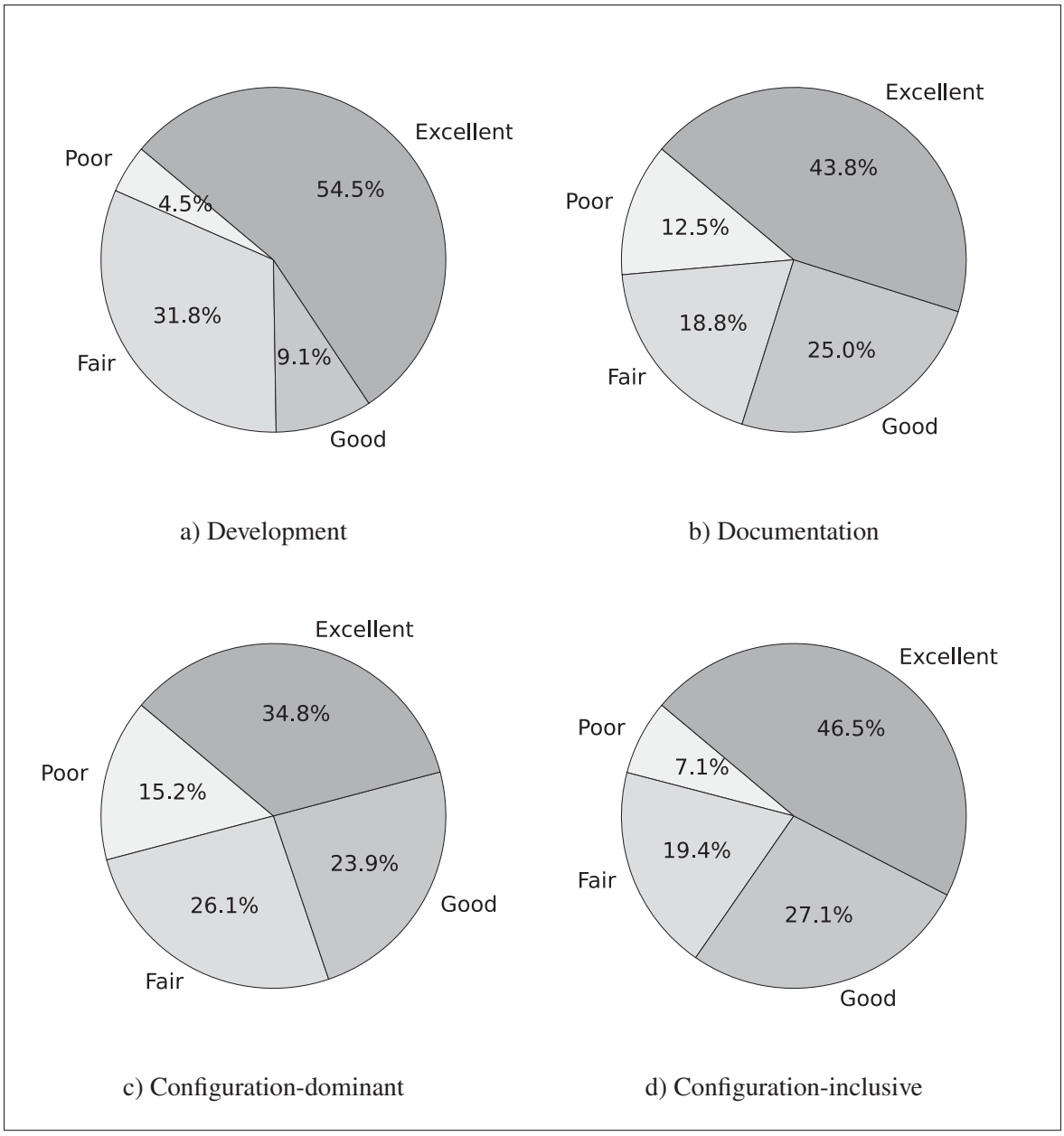


Figure 6.2 Distribution of review grades for different types of MRs

To better understand *how* review feedback differs, Figure 6.3 compares the presence of review attributes. Development reviews frequently include *Suggestion* (52.4%) and *Evaluation* (42.9%), and less frequently include *Question* (19.0%). Configuration-inclusive reviews closely mirror this structure: *Suggestion* appears in 52.3% of reviews and *Evaluation* in 49.7%, while *Question* is slightly higher (22.6%). By contrast, configuration-dominant reviews show a markedly

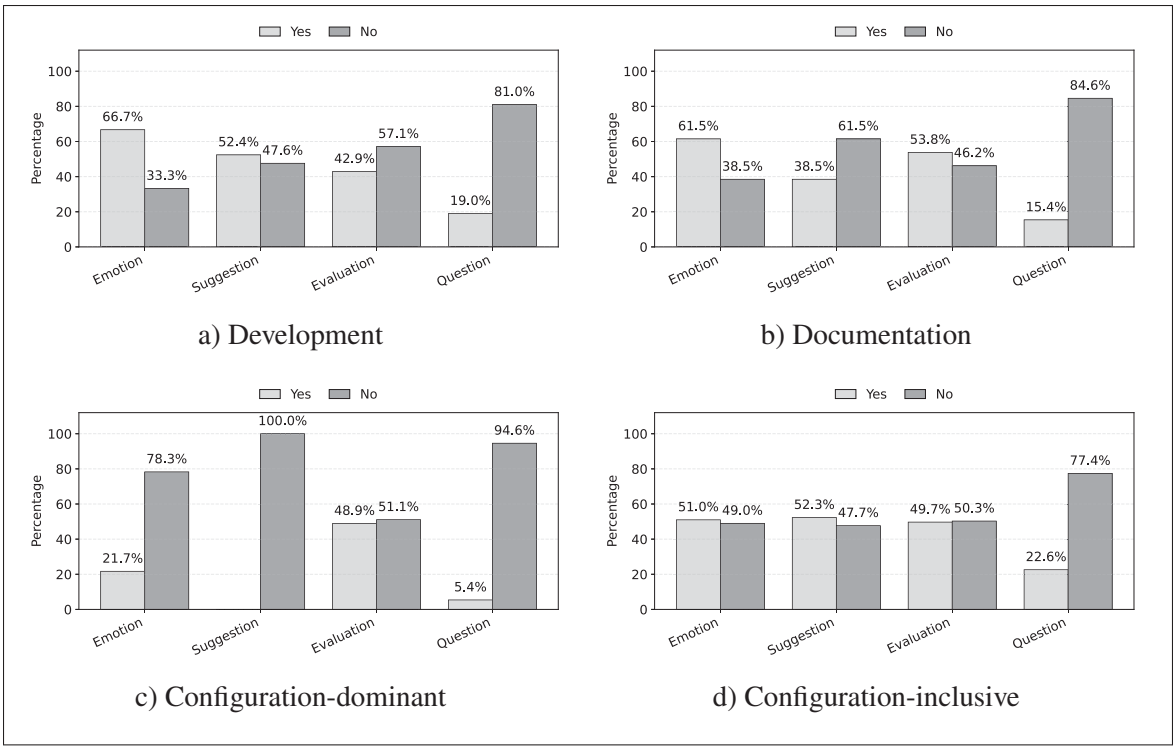


Figure 6.3 Distribution of review attributes for different types of MRs

reduced presence of interactive and improvement-oriented feedback: *Question* is rare (5.4%), and *Suggestion* is effectively absent in our coded sample (Figure 6.3c), while *Evaluation* remains moderate (48.9%). This pattern indicates that configuration-dominant reviews are more often limited to validation/assessment, with fewer clarifying exchanges and fewer concrete improvement proposals.

To address the risk of conflating configuration and development feedback in mixed MRs, we further inspected the annotated review threads for configuration-inclusive MRs and assessed whether comments target configuration changes or non-configuration (development) changes. We selected a representative sample of configuration-inclusive MRs to analyse similarly to the steps followed in 6.2.4.2. We found that approximately 71% of the review comments in configuration-inclusive MRs are oriented toward the development part of the change, with limited feedback on the configuration portion. This finding suggests that the higher observed

review grades for configuration-inclusive MRs may partially reflect well-established code-review practices applied to development changes, rather than configuration-specific scrutiny.

6.3.2.2 Configuration vs. Documentation

Documentation reviews (Figure 6.2) show intermediate quality compared to the two configuration categories: 43.8% of reviews are rated *Excellent* and 25.0% *Good*, while 18.8% are *Fair* and 12.5% *Poor*. Configuration-dominant reviews remain less favorable than documentation reviews, with a higher *Poor* share (15.2% vs. 12.5%) and a lower combined *Good+Excellent* proportion (58.7% vs. 68.8%). Configuration-inclusive reviews are rated more positively than documentation reviews overall (73.6% *Good+Excellent* vs. 68.8%) and show fewer *Poor* reviews (7.1% vs. 12.5%).

The attribute distributions in Figure 6.3 help clarify these differences. Documentation reviews include *Suggestion* in 38.5% of cases and *Evaluation* in 53.8%, while *Question* remains relatively limited (15.4%). Configuration-dominant reviews are comparable in *Evaluation* (48.9%) but exhibit substantially fewer *Questions* (5.4%) and near-absent *Suggestions*, suggesting less interactive review exchanges and fewer concrete proposals. Configuration-inclusive reviews again resemble a development-like feedback profile, with higher *Suggestion* (52.3%) than documentation and a similar level of *Evaluation* (49.7%), while maintaining a relatively higher presence of *Questions* (22.6%). Taken together, these results indicate that documentation reviews provide structured assessment, but configuration-dominant reviews are the most likely to lack actionable suggestions and clarifying dialogue.

To ground these patterns concretely, we provide representative review excerpts for each MR type (anonymized and rephrased to preserve confidentiality). A typical development review (often rated *Good/Excellent*, which together account for 63.6% of development grades) includes actionable, improvement-oriented feedback, e.g., *"Please add a unit test for the timeout branch and handle the null case explicitly to avoid silent failures."* (Suggestion/Evaluation). This aligns with the high *Suggestion* presence (52.4%) observed in development reviews. In contrast, a

typical configuration-dominant review—where *Fair* and *Poor* grades are more prevalent than in any other MR type (26.1% and 15.2%, respectively)—is validation-oriented and less interactive, e.g., *"This update changes rule precedence and will alter the effective traffic match for this policy."* (Evaluation only), consistent with the near-zero *Suggestion* and 5.4% *Question* rates for this category. Configuration-inclusive reviews are frequently rated *Good/Excellent* (73.6% combined) and exhibit attribute distributions similar to development reviews. However, as reported above, approximately 71% of their review comments target the development portion of the change, e.g., *"Add an integration test for the new controller logic; ensure defaults remain unchanged."* (Suggestion/Evaluation; dev-focused). Documentation reviews offer structured but lightweight guidance, with moderate *Suggestion* (38.5%) and higher *Evaluation* (53.8%), e.g., *"Please align terminology with the rest of the guide and add a short command/output example for verification."* (Suggestion/Evaluation).

6.3.3 Quantitative analysis of bug vs. non-bug configuration MRs

RQ3: What are the differences in the review process between bug-related and non-bug-related configuration MRs

To answer RQ3, we split configuration-related MRs into bug-associated and non-bug MRs using the bug-identification workflow described in Section 6.2.2.2 (labels when available, complemented by validated heuristics). We then repeat the same quantitative comparisons as in RQ1 on the two subsets (bug vs non-bug), using the same metrics, statistical tests, and effect sizes to ensure consistency.

Configuration bug-related MRs exhibit significantly higher review activity than configuration non-bug MRs (p-value <0.001). As shown in Table 6.6, both configuration-dominant and configuration-inclusive bug-related MRs have significantly lower initial MR size and additions than their non-bug counterparts, while exhibiting significantly higher deletions. This pattern is consistent with bug-fixing work, which often focuses on removing or reverting problematic logic rather than introducing new functionality. Importantly, *both* configuration-dominant and

Table 6.6 One-tailed Mann–Whitney U test results comparing bug-related and non-bug-related configuration MRs

Dimension	Metric	config-d	config-i
Activity	#Comments	*non-bug* < *bug* ***	non-bug < bug ***
	#Reviewer's comments	non-bug < bug ***	*non-bug* < *bug* ***
	MR size	non-bug > bug ***	non-bug > bug ***
	#Additions	non-bug > bug ***	non-bug > bug ***
	#Deletions	non-bug < bug ***	*non-bug* < *bug* ***
	Rework size	*non-bug* < *bug* ***	non-bug < bug ***
Engagement	#Committers	non-bug < bug ***	*non-bug* < *bug* ***
	#Reviewers	non-bug < bug ***	*non-bug* < *bug* ***
	#Collaborators	non-bug < bug *	*non-bug* < *bug* ***
Timeline	Time to complete peer review	*non-bug* < *bug* ***	*non-bug* < *bug* ***
	Time to first review	non-bug < bug **	*non-bug* < *bug* ***
Approval	Approved	non-bug < bug ***	*non-bug* < *bug* ***
	Self-Approved	non-bug < bug ***	non-bug < bug ***
	Self-Merged	non-bug > bug *	non-bug > bug
	Self-Assigned	non-bug < bug **	*non-bug* < *bug* ***

config-d: configuration-dominant; config-i: configuration-inclusive.
Statistical significance: * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$.
Effect size: normal = negligible, *italic* = small, underline = medium, **bold** = large.

configuration-inclusive bug-related MRs receive significantly more comments overall and more reviewer comments in particular. They also require significantly more rework during review, indicating that addressing defects typically involves additional iterations to refine the fix and ensure correctness.

A plausible explanation is that bug reports provide an explicit problem statement and a concrete failure scenario, which encourages reviewers to (i) validate the root cause, (ii) scrutinize whether the proposed changes fully resolve the issue, and (iii) reason about potential side effects in configuration-sensitive runtime behavior. This naturally increases discussion volume and drives additional rework before approval.

As shown in Table 6.6, configuration bug-related MRs tend to have significantly higher review engagement compared to configuration non-bug-related MRs. For both configuration-dominant and configuration-inclusive MRs, the presence of a bug leads to significantly increased engagement from a greater number of committers (p-value<0.001), indicating that bugs prompt a higher level of collaborative effort to resolve issues. Similarly, the higher number of reviewers for both types of configuration bug-related MRs suggests that bugs necessitate more thorough scrutiny from diverse perspectives (p-value<0.001). Additionally, the increased number of collaborators involved in discussions (p-value<0.05) reflects the broader range of expertise required to effectively handle and review bug-related issues. Bug-related work is often triage-driven and cross-cutting; it tends to mobilize additional contributors and reviewers because diagnosing and validating fixes may require operational knowledge, historical context, and independent confirmation before deployment.

Configuration bug-related MRs take longer to complete peer review compared to configuration non-bug-related MRs. Table 6.6 shows that both configuration-dominant bug-related and configuration-inclusive bug-related MRs require significantly more time to get the first review compared to non-bug-related MRs (p-value <0.01). This longer first-review time suggests that collaborators need more time to understand the bug context before commenting, which can also contribute to increased time to complete peer review Hasan *et al.* (2023). Effectively, these MRs also take longer to complete peer review, aligning with our previous observations. These findings indicate that configuration bug-related MRs often involve more discussions, collaborations, and modifications, resulting in increased effort and extended time to complete peer review. Compared to routine changes, bug fixes typically require additional sensemaking (reproducing/understanding the failure) and stronger validation, which delays first feedback and extends completion due to iterative confirmation and risk mitigation.

The analysis of the approval dimension, as shown in Table 6.6, reveals that configuration bug-related MRs are significantly more likely to be approved compared to configuration non-bug-related MRs. Specifically, both configuration-dominant and configuration-inclusive bug-related MRs tend to receive significantly more approvals—including self-approvals and

self-assignments—when compared to non-bug-related MRs. This suggests that the presence of a bug increases the urgency and perceived importance of the MR, leading to higher rates of approval. However, these bug-related configuration MRs are less likely to be self-merged, especially configuration-dominant MRs (p-value<0.05), indicating that although authors may approve their own fixes, the final merge is more often performed by another collaborator. Bug-related configuration changes may follow stricter release discipline: approvals increase due to urgency, while self-merging decreases because teams may require an independent integrator to reduce risk before deploying a bug fix that can affect production behavior.

6.3.4 Qualitative analysis of bug vs. non-bug configuration MRs

RQ4: How does the quality of reviews differ between bug-related and non-bug-related configuration MRs

To answer RQ4, we compare the qualitative review outcomes between bug-associated and non-bug configuration MRs using the same annotation scheme as in RQ2. Specifically, we analyze differences in the distribution of review quality grades (Poor to Excellent) and the presence of review attributes (Emotion, Question, Evaluation, Suggestion) across the two bug-related subsets. The qualitative framework and coding procedure are described in Section 6.2.4.2.

Interestingly, configuration-dominant bug-related MRs exhibit a remarkable 79% of reviews rated as good or excellent, with no poor reviews at all. As shown in Figure 6.4, in configuration-dominant non-bug-related MRs (Figure 6.4a), the reviews are relatively mixed, with 36.4% rated as excellent, 25.0% as good, 22.7% as fair, and 15.9% as poor. This indicates that while the majority of reviews are positive, there is still a significant portion that reflects fair to poor quality, suggesting some inconsistency in the review process. In contrast, configuration-dominant MRs that are related to a bug (Figure 6.4b) show a much more positive distribution. A notable 57.9% of reviews are rated as excellent, with an equal 21.1% rated as good and fair. In addition, none of these reviews are rated as poor. This stark improvement in review grades suggests that the

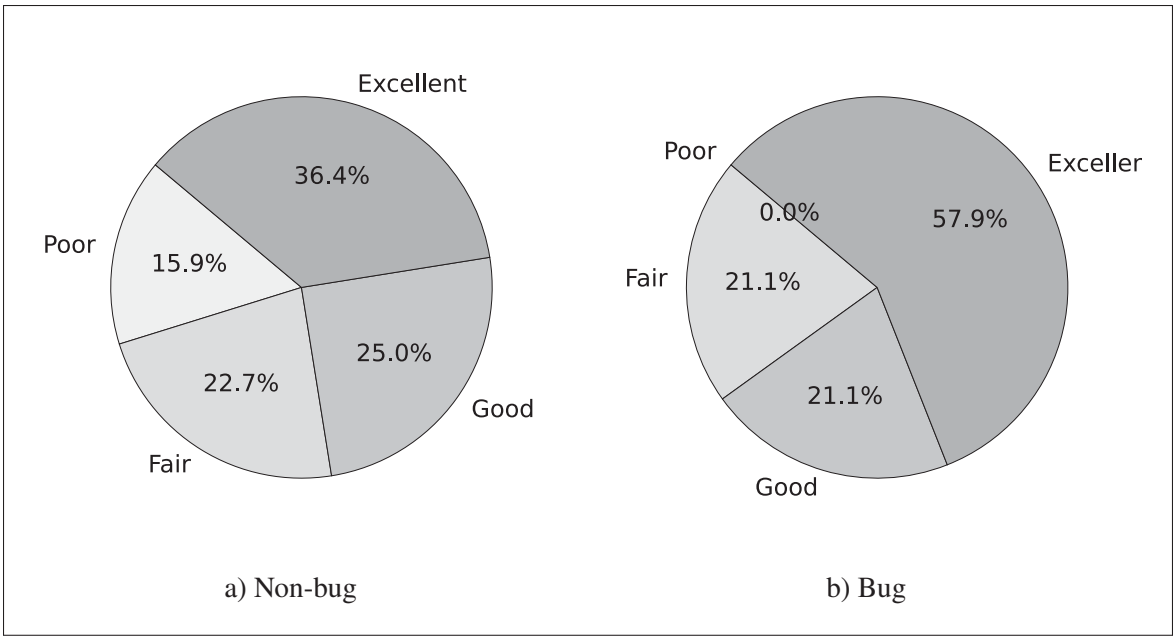


Figure 6.4 Distribution of review grades for configuration-dominant bug-related and non-bug-related MRs

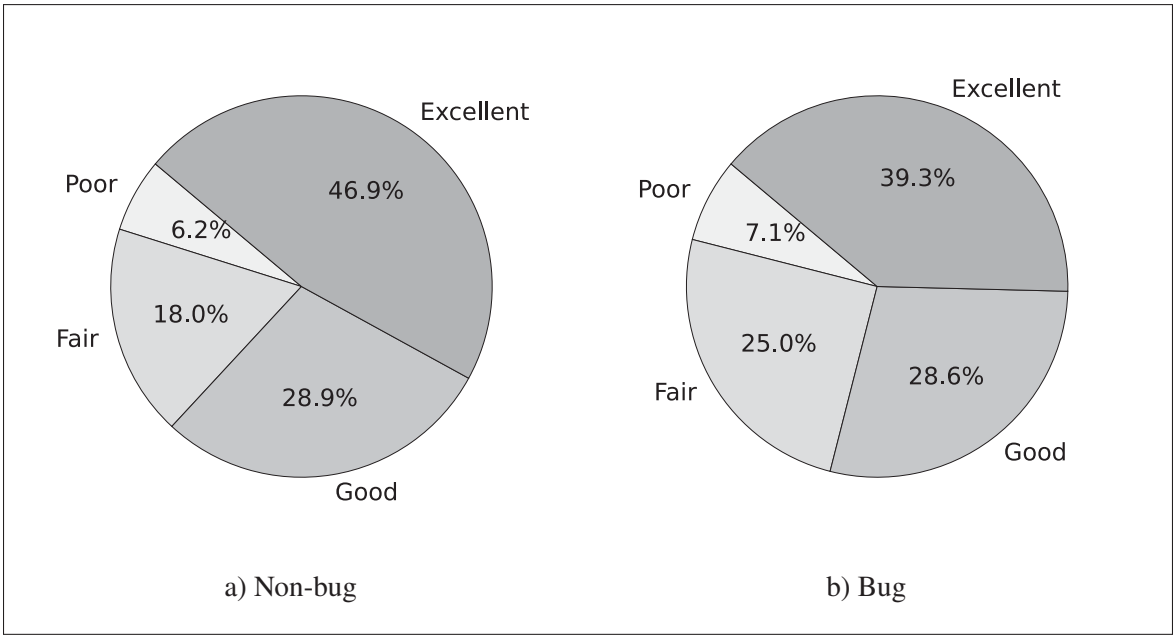


Figure 6.5 Distribution of review grades for configuration-inclusive bug-related and non-bug-related MRs

presence of a bug in the configuration MR may lead to more thorough and higher-quality reviews, as reviewers likely focus more intently on identifying and resolving the issue.

Configuration-dominant bug-related MRs lead to more thorough evaluations and a stronger focus on technical accuracy, compared to the more balanced but less focused reviews of non-bug-related MRs. As shown in Figure 6.6, in non-bug-related MRs (Figure 6.6a), there is a balanced distribution across most attributes, with emotions and evaluations being the most prominent. Specifically, 56.8% of reviews include emotions, and 56.8% focus on evaluations, reflecting a mix of technical assessment and personal feedback, potentially due to less urgency or criticality in the review. However, suggestions and questions are less frequently included, with only 36.4% of reviews offering suggestions and 18.2% raising clarifying questions. This could indicate that reviews are less focused on improvement and understanding when no bugs are present. In contrast, bug-related MRs (Figure 6.6b) demonstrate a stronger emphasis on evaluations and questions. A significant 78.9% of reviews include evaluations, highlighting the reviewers' increased scrutiny of the code's functionality to ensure bugs are properly addressed. The inclusion of questions rises to 26.3%, indicating a greater need for clarification and deeper understanding in the context of bug-related reviews. Suggestions are present in 42.1% of these reviews, showing increased efforts to provide constructive feedback in the presence of a bug. Interestingly, emotion-based comments drop to 36.8% in bug-related MRs, suggesting that reviewers may prioritize technical accuracy and problem-solving over emotional tone when addressing bugs, reflecting the seriousness of the task.

To validate that our "bug" labels correspond to truly bug-fixing discussions, we manually inspected the review comments and estimated the share that explicitly refers to corrective actions (e.g., fixing incorrect behavior, resolving errors, addressing failures, or proposing a concrete patch). Using this criterion, approximately 65% of configuration-inclusive comments and 62% of configuration-dominant comments were clearly bug-fix oriented. This provides additional confidence that the MRs labeled as bug-related are not merely tagged as "bug" administratively, but are predominantly discussed as defect-resolution work in the review threads, supporting the validity of our bug/non-bug categorization. Importantly, this qualitative confirmation aligns

with our quantitative results: when reviews explicitly focus on bug resolution, they also tend to be more rigorous and consistently higher-quality, reinforcing our observation that review quality improves in the presence of bugs.

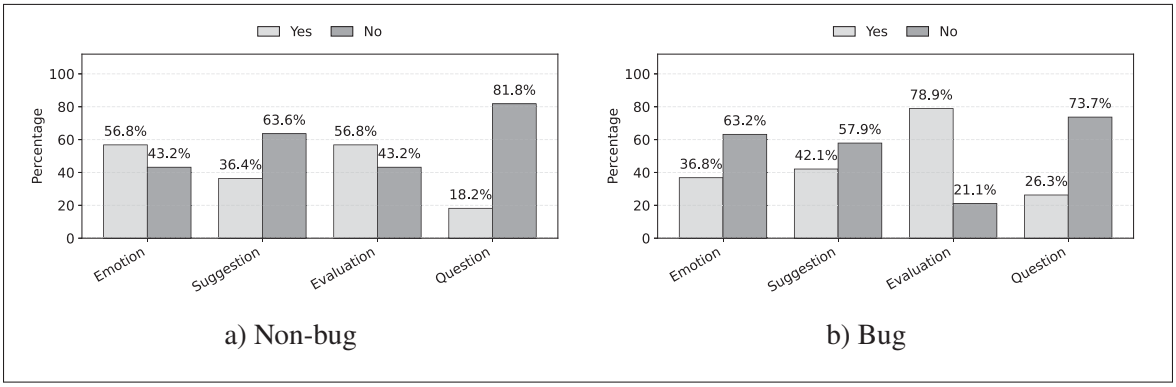


Figure 6.6 Distribution of review attributes for configuration-dominant bug-related and non-bug-related MRs

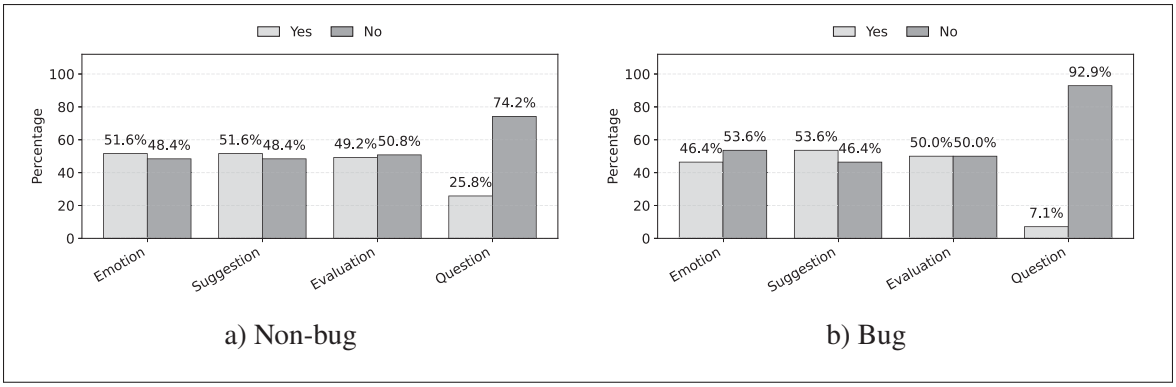


Figure 6.7 Distribution of review attributes for configuration-inclusive bug-related and non-bug-related MRs

The analysis of review grades highlights similarities in review quality between configuration-inclusive MRs related and non-related to bugs. When no bugs are present (Figure 6.5a), the reviews are predominantly well done, with 46.9% rated as excellent and 28.9% as good, resulting in a combined 75.8% of reviews being favorable. However, there are still some challenges, with 18% of reviews rated as fair and a smaller portion, 6.2%, rated as poor. This suggests that, in the absence of bugs, configuration-inclusive MRs tend to receive higher quality reviews, although there is still room for improvement. When bugs are involved (Figure 6.5b), the review

distribution shifts slightly but remains largely similar to non-bug-related MRs. The proportion of excellent reviews decreases slightly to 39.3%, while good reviews remain steady at 28.6%. The percentage of fair reviews increases to 25%, and poor reviews see a modest rise to 7.1%. This slight shift indicates that configuration-inclusive MRs tend to receive strong feedback even when bugs are present, though the review process becomes slightly more challenging. Overall, the reviews consistently fall within the good or excellent categories, demonstrating robust review quality despite the added complexity of bugs.

As shown in Figure 6.7, for configuration-inclusive MRs, the distribution across all attributes remains consistent regardless of the presence of a bug, except the question attribute. For both bug-related and non-bug-related MRs, the suggestion and evaluation attributes are nearly equal, with 51.6% and 49.2% for suggestions and evaluations in non-bug-related MRs, and 53.6% and 50% for bug-related MRs, respectively. This suggests that the presence of a bug does not significantly influence the focus on these attributes, possibly because both types of changes demand careful assessment and constructive feedback, irrespective of the bug's presence. Similarly, the emotion attribute shows close values between non-bug-related and bug-related MRs, with 51.6% and 46.4%, respectively. This indicates that reviewers maintain a similar level of emotional engagement, whether a bug is present. However, a notable difference is observed in the question attribute. Surprisingly, the number of questions decreases sharply from 25.8% in non-bug-related MRs to just 7.1% in bug-related MRs. This suggests that the presence of a bug may lead reviewers to focus more on understanding how the issue at hand has been resolved rather than seeking additional clarification in this type of MRs, especially when MRs involve many types of changes.

To illustrate the patterns concretely, we provide representative review excerpts for each subset. In configuration-dominant bug-related MRs, where 79% of reviews are rated good or excellent, comments tend to be technically focused and grounded in validation, e.g., *"This change fixes the incorrect match condition; with the updated rule precedence, the policy now selects the expected traffic class."* (Evaluation). In configuration-dominant non-bug-related MRs, where review quality is more variable (15.9% rated poor, compared to 0% for bug-related),

Table 6.7 Overview of the number of MRs per category for Omnibus

Category	Config		Non-Config	
	config-d	config-i	dev	doc
Total MRs	1,573 (44%)	1,939 (56%)	1,190 (49%)	1,232 (51%)
Bug-related MRs	786 (49%)	917 (47%)	–	–
Overall	3,512 (64%)		2,422 (36%)	

config-d: configuration-dominant; config-i: configuration-inclusive; dev: development; doc: documentation.

comments are more likely to lack explicit validation criteria, e.g., *"The update changes the policy behavior, but the discussion does not clarify the expected outcome or the validation evidence."* (Evaluation). For configuration-inclusive MRs, review attributes remain stable regardless of bug presence, and comments often retain a development-oriented character, e.g., *"The fix looks correct; please add an integration test for the regression path while keeping defaults unchanged."* (Suggestion/Evaluation). This consistency aligns with the stable distribution of suggestions and evaluations observed for this subset, although the notable decrease in questions for bug-related configuration-inclusive MRs (from 25.8% to 7.1%) suggests that reviewers shift from seeking clarification to validating the resolution when a known defect is present.

6.4 Discussion

To evaluate whether our findings extend beyond the SDN and industrial context, we replicated our methodology on two open-source projects: **GitLab Omnibus**[1] (the package manager and deployment toolchain for GitLab, rich in configuration files), and **GitLab Runner**[2] (an execution agent for GitLab CI/CD pipelines, heavily relying on configuration and automation scripts). Since both projects show similar trends, we focus our discussion on the results from Omnibus, summarized in Table 6.7. Detailed results for GitLab Runner are available in our replication package.

[1] https://gitlab.com/gitlab-org/omnibus-gitlab

[2] https://gitlab.com/gitlab-org/gitlab-runner

Table 6.8 Omnibus: Differences in review activity, engagement, timeline, and self-management between configuration and development MRs (Mann–Whitney U, one-tailed)

Dimension	Metric	config-d vs. dev	config-i vs. dev
Activity	#Comments	config-d < dev *	*config-i > dev* ***
	#Reviewer's comments	config-d > dev	config-i > dev ***
	MR size	*config-d < dev* ***	config-i > dev ***
	#Additions	*config-d < dev* ***	config-i > dev ***
	#Deletions	*config-d < dev* ***	*config-i > dev* ***
	Rework size	config-d < dev *	*config-i > dev* ***
Engagement	#Committers	config-d > dev	config-i > dev ***
	#Reviewers	config-d < dev ***	config-i < dev *
	#Collaborators	config-d < dev **	config-i > dev **
Timeline	Time to complete peer review	config-d < dev ***	*config-i > dev* ***
	Time to first review	config-d < dev ***	config-i > dev ***
Approval	Approved	*config-d > dev*	config-i < dev
	Self-Approved	config-d > dev ***	config-i > dev **
	Self-Merged	config-d > dev **	config-i > dev **
	Self-Assigned	config-d > dev ***	config-i > dev ***

config-d: configuration-dominant; config-i: configuration-inclusive; dev: development.
Statistical significance: * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$.
Effect size: normal = negligible, *italic* = small, underline = medium, **bold** = large.

Comparison with Development MRs. Table 6.8 shows both converging and diverging patterns compared to TELUS. Consistent with our industrial results, configuration-inclusive MRs in Omnibus exhibit significantly higher review activity and rework than development MRs. The same direction also holds for initial MR size and additions, whereas deletions differ: development MRs contain more deletions than configuration-dominant MRs. This reaffirms that the complexity introduced by mixed-type changes tends to induce deeper reviewer engagement. However, configuration-dominant MRs in Omnibus show less consistency in engagement. While they receive fewer comments and involve less rework (similar to TELUS), they sometimes elicit more reviewer comments than expected, potentially reflecting different norms or expectations in open-source review culture.

Table 6.9 Omnibus: Differences in review activity, engagement, timeline, and self-management between configuration and documentation MRs (Mann–Whitney U, one-tailed)

Dimension	Metric	config-d vs. doc	config-i vs. doc
Activity	#Comments	*config-d < doc* **	config-i > doc ***
	#Reviewer's comments	config-d > doc	config-i < doc *
	MR size	config-d < doc ***	config-i > doc ***
	#Additions	config-d < doc ***	config-i > doc ***
	#Deletions	config-d > doc ***	config-i > doc ***
	Rework size	config-d < doc ***	*config-i > doc* ***
Engagement	#Committers	config-d > doc	config-i > doc ***
	#Reviewers	config-d > doc	config-i < doc ***
	#Collaborators	config-d < doc ***	*config-i > doc*
Timeline	Time to complete peer review	config-d > doc **	*config-i > doc* ***
	Time to first review	config-d > doc **	config-i > doc ***
Approval	Approved	config-d > doc	config-i < doc *
	Self-Approved	config-d > doc *	config-i < doc *
	Self-Merged	config-d > doc **	config-i > doc *
	Self-Assigned	config-d > doc ***	config-i > doc ***

config-d: configuration-dominant; config-i: configuration-inclusive; doc: documentation.
Statistical significance: * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$.
Effect size: normal = negligible, *italic* = small, underline = medium, **bold** = large.

Comparison with Documentation MRs. As shown in Table 6.9, we observe the same directional patterns for initial MR size, additions, and deletions. Configuration-inclusive MRs again exhibit significantly higher review activity than documentation MRs, reinforcing that mixed changes (configuration combined with code and/or tests) consistently attract more scrutiny and iterative feedback than documentation-only updates, regardless of project context. Interestingly, configuration-dominant MRs in Omnibus are not always marginalized in reviews, as they were at TELUS. In some cases, they attract comparable or even more attention than documentation MRs in terms of time to complete peer review or reviewer count. This could be attributed to the lack of formalized review roles and more distributed ownership in open-source development, which can blur traditional hierarchies in review participation.

Comparison of Bug and Non-Bug Configuration MRs. As shown in Table 6.10, we observe the same patterns as in TELUS for initial MR size, additions, and deletions. However, in Omnibus, bug-related configuration MRs tend to involve *less* rework and lower committer engagement than non-bug MRs. This contrast is most pronounced for configuration-inclusive MRs, where non-bug MRs attract more collaborators and trigger more extensive revisions. A plausible explanation is that many non-bug, mixed-type changes align with ongoing feature work or broader refactoring and integration efforts, which naturally invite more collective iteration. In comparison, bug-fixing MRs are often narrower in scope, handled by fewer contributors, and aimed at quickly restoring correct behavior, which can limit both collaboration and rework.

Table 6.10 Omnibus: Differences between bug-related and non-bug configuration MRs in review activity, engagement, timeline, and self-management (Mann–Whitney U, one-tailed)

Dimension	**Metric**	**config-d**	**config-i**
Activity	#Comments	non-bug < bug ***	non-bug < bug ***
	#Reviewer's comments	non-bug < bug ***	non-bug < bug ***
	MR size	non-bug > bug ***	*non-bug > bug* ***
	#Additions	non-bug > bug ***	non-bug > bug ***
	#Deletions	non-bug < bug ***	non-bug < bug ***
	Rework size	non-bug > bug ***	non-bug > bug ***
Engagement	#Committers	non-bug > bug ***	non-bug > bug ***
	#Reviewers	non-bug < bug ***	non-bug < bug ***
	#Collaborators	non-bug < bug *	non-bug < bug ***
Timeline	Time to complete peer review	non-bug < bug ***	non-bug < bug ***
	Time to first review	non-bug < bug **	non-bug < bug ***
Approval	Approved	non-bug < bug ***	non-bug < bug ***
	Self-Approved	non-bug > bug ***	non-bug > bug ***
	Self-Merged	non-bug < bug *	non-bug > bug
	Self-Assigned	non-bug < bug **	non-bug < bug ***

config-d: configuration-dominant; config-i: configuration-inclusive.
Statistical significance: * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$.
Effect size: normal = negligible, *italic* = small, underline = medium, **bold** = large.

Review Quality Comparison. Beyond activity metrics, we analyze the quality of review comments across project types, as shown in Figures 6.8 and 6.9. Omnibus results reinforce

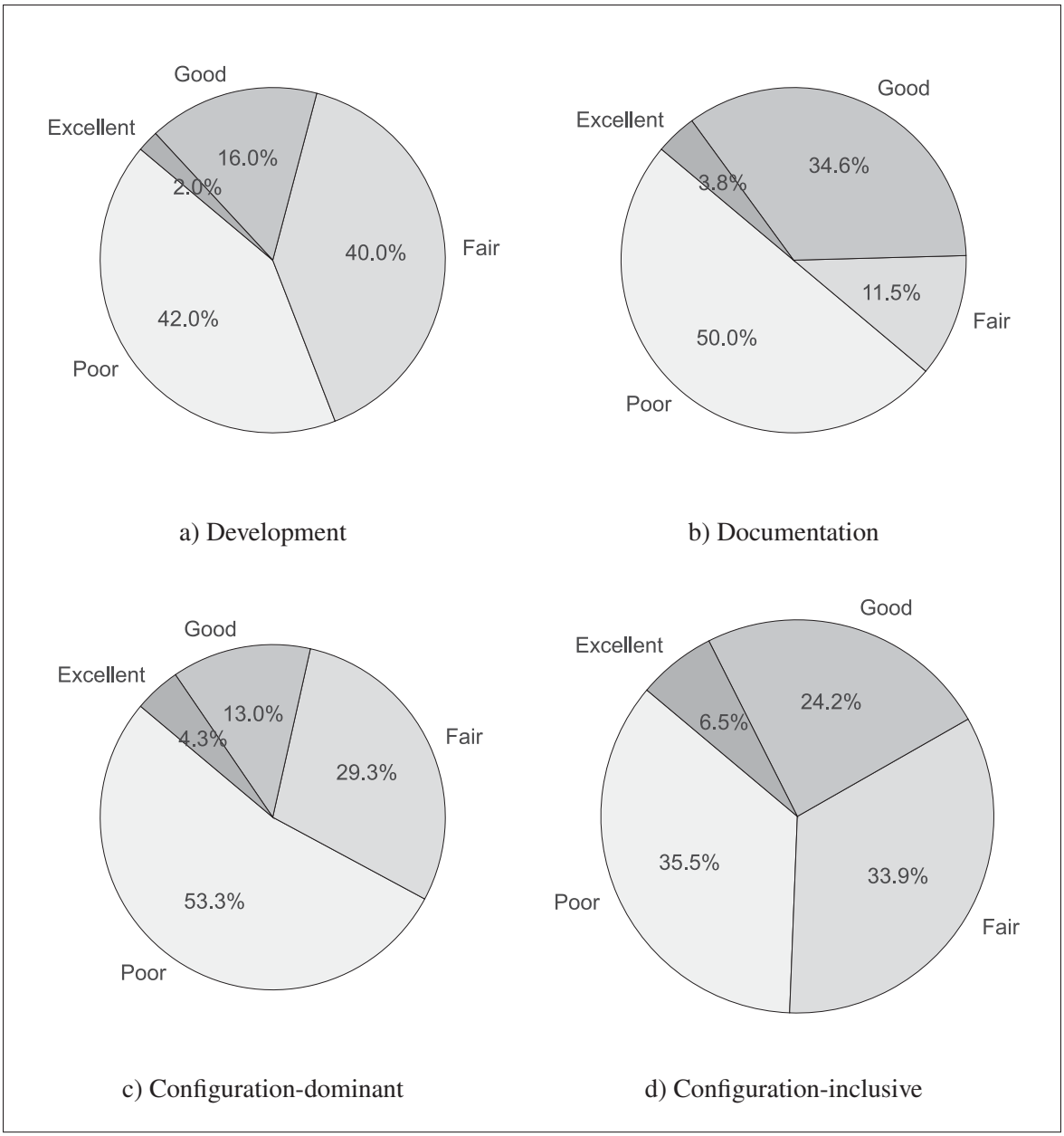


Figure 6.8 Omnibus: Distribution of review quality grades across MR types (development, documentation, configuration-dominant, configuration-inclusive)

TELUS trends: configuration-dominant and documentation MRs are more frequently rated as *Poor* or *Fair*, while configuration-inclusive MRs show a more balanced quality distribution. Notably, bug-related MRs, although receiving more review engagement, do not consistently exhibit higher quality. As shown in Figures 6.10 and 6.11, for configuration-dominant bug-related

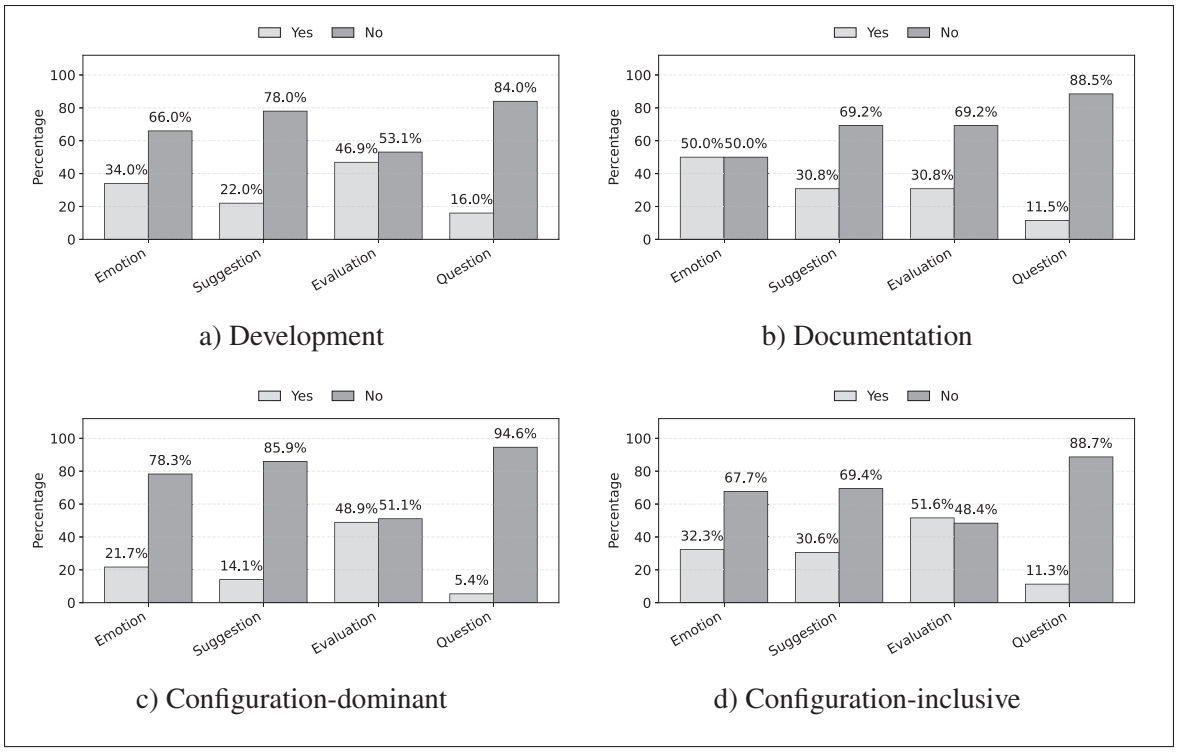


Figure 6.9 Omnibus: Distribution of review attributes (Emotion, Question, Evaluation, Suggestion) across MR types

MRs in Omnibus, Poor grades still dominate (58.3%), and Excellent grades are absent. In contrast, configuration-inclusive bug-related MRs display slightly better quality, with higher proportions of Fair and Excellent grades.

Moreover, Figures 6.12 and 6.13 show that review attributes like suggestion and *q*uestion are more frequent in bug-related MRs, especially in configuration-inclusive ones, reaffirming that bug fixes stimulate deeper review reasoning. These results also reflect a more interactive feedback style in open-source settings (e.g., Omnibus), where Question attributes are more prevalent compared to TELUS. This suggests differences in communication dynamics—open-source reviewers may engage more dialogically, while industry reviews lean toward formal evaluation.

General Trends. Across both comparisons, self-management remains a dominant theme in Omnibus as in TELUS. Configuration-dominant MRs are more likely to be self-approved,

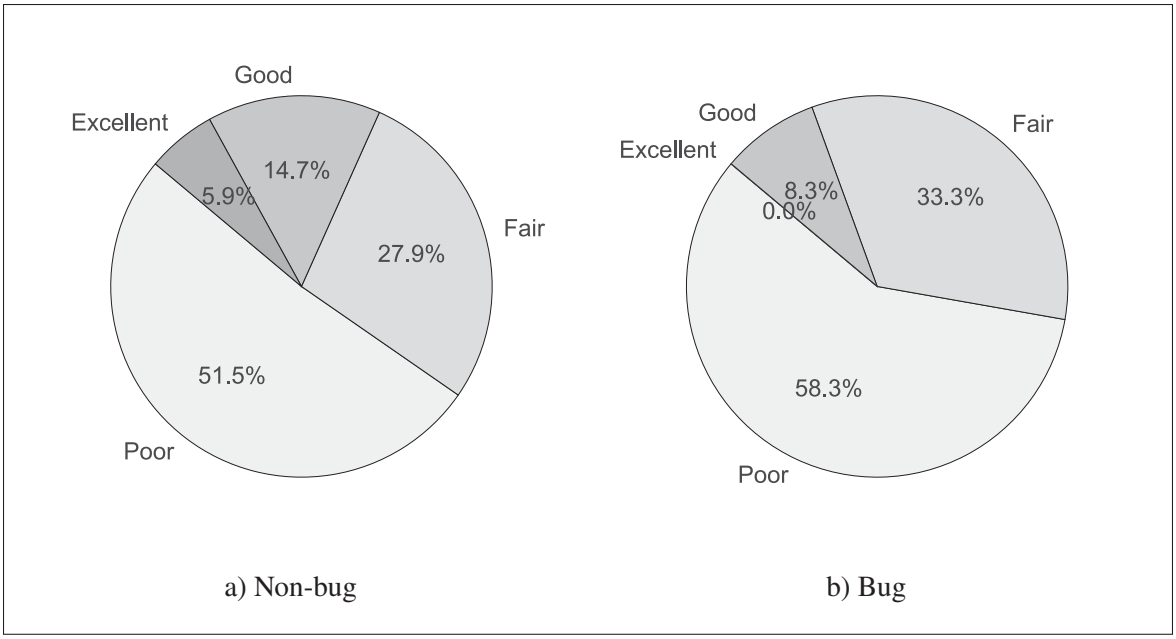


Figure 6.10 Omnibus: Review quality grades for bug-related vs. non-bug configuration-dominant MRs

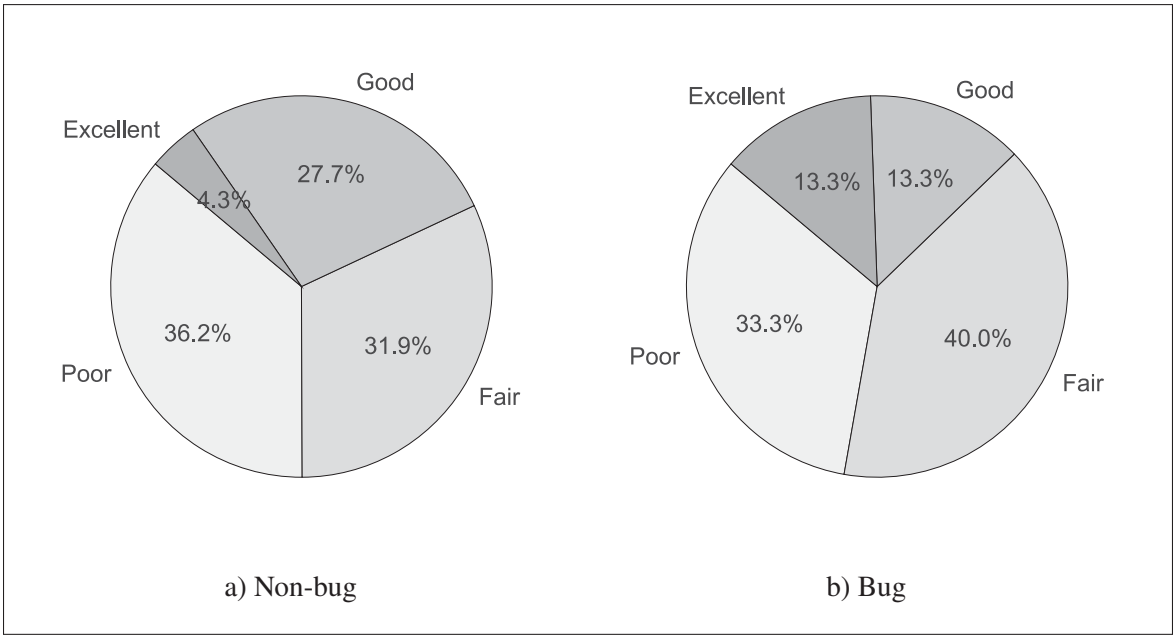


Figure 6.11 Omnibus: Review quality grades for bug-related vs. non-bug configuration-inclusive MRs

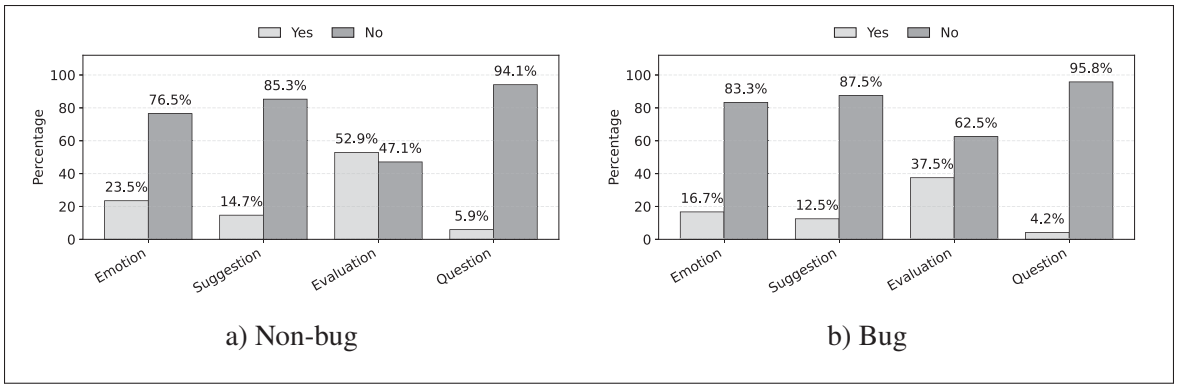


Figure 6.12 Omnibus: Review attributes for bug-related vs. non-bug configuration-dominant MRs

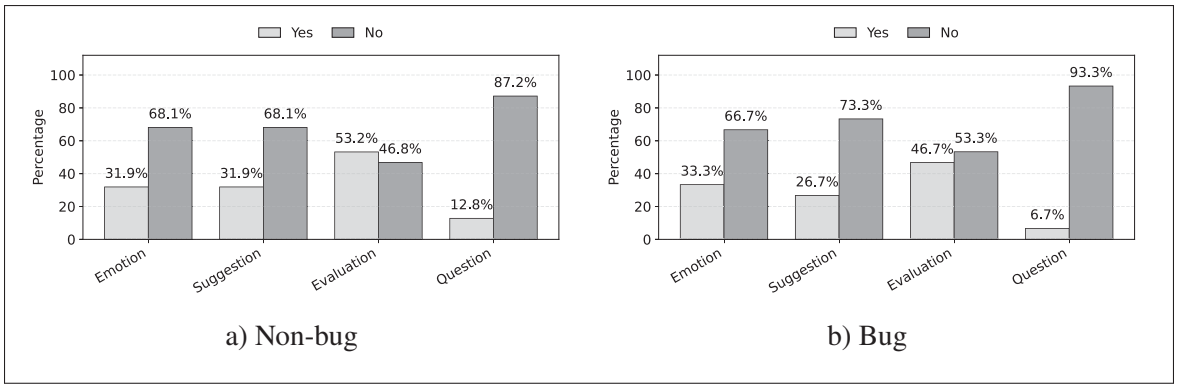


Figure 6.13 Omnibus: Review attributes for bug-related vs. non-bug configuration-inclusive MRs

self-merged, and self-assigned, suggesting a broader trend where contributors with expertise over configuration components take greater ownership, regardless of project type.

Overall, these findings suggest that while the exact level of review attention to configuration MRs may vary across contexts, the fundamental dynamics—such as the link between complexity and engagement, quality limitations in configuration review, or the self-managed nature of configuration work—persist across industrial and open-source settings. Omnibus and GitLab Runner provide meaningful contrasts that strengthen the external validity of our methodology. Their diversity of contributors, lack of formal review roles, and community-driven development

practices highlight how organizational structure can modulate review behaviors, while still reflecting shared challenges around reviewing configuration-intensive changes.

6.5 Implications

6.5.1 Implications for our industrial partners

Our study reveals important patterns in how configuration-related MRs are reviewed compared to development and documentation MRs, especially in the context of SDN systems. This section translates these patterns into actionable recommendations for our industrial partners (TELUS in particular) and, by extension, for practitioners in similar configuration-intensive environments.

1. Enforce peer review for configuration-dominant changes (policy + tooling). Configuration-dominant MRs receive fewer reviewer comments and are often self-approved, indicating limited external scrutiny. TELUS can operationalize this by (i) defining a *CODEOWNERS*-style ownership rule for high-impact configuration paths (e.g., policy/ACL/service-chain/controller templates) to automatically request at least one independent reviewer, and (ii) adding a merge policy in GitLab that blocks merging when a configuration file is touched unless the required approvals are obtained. This ensures that even small diffs in semantics-bearing configuration receive external validation.

2. Increase reviewer diversity via routing rules (process + training). We observed lower reviewer engagement and higher self-management in configuration MRs, despite configuration work often requiring cross-team expertise (network, infrastructure, security). TELUS can implement reviewer routing by mapping configuration folders to reviewer pools (e.g., networking for routing/policies, security for ACLs/segmentation, platform for templates/orchestration) and using automatic reviewer assignment. A short internal training or "review playbook" can further align expectations by illustrating common misconfiguration failure modes and what reviewers should verify.

3. Establish configuration-specific review checklists and validation gates (standardization + automation). To reduce ad-hoc reviewing, TELUS can attach a configuration checklist to MRs that touch configuration (e.g., verify policy precedence/defaults, confirm intended blast radius, ensure staging validation evidence, check backward compatibility of templates). This can be operationalized by (i) MR templates that require authors to fill "intent + impact + validation" fields, and (ii) CI jobs that run configuration linters/validators or policy tests when relevant files change, providing reviewers with objective signals beyond the diff.

4. Monitor review effectiveness beyond activity metrics (measurement + feedback loop). Larger configuration-inclusive MRs attract more comments and reviewers, but higher activity does not always imply higher review quality. TELUS can operationalize effectiveness monitoring by tracking outcome-oriented indicators such as post-merge incidents, rollbacks, hotfixes, or reopened MRs, and linking them to MR characteristics (e.g., self-managed vs. externally reviewed, config-d vs. config-i). This enables managers to identify where reviews are active but ineffective, and to target policy updates or coaching accordingly.

5. Transferable tooling for configuration recognition. Many repositories lack explicit labelling of configuration-related MRs, which could nevertheless help improve the review process by enabling teams to route such MRs through tailored review workflows and assign specialized reviewers. Our heuristic-based classification—leveraging file extensions, folder names, and file-type ratios—can be integrated into CI/CD pipelines, for example, to automatically detect configuration-heavy MRs. The same pipeline can be reused by any industrial partner that operates on Git-based review platforms.

6. Recognizing the growing role of configuration. Configuration is no longer a peripheral concern; it is integral to system behavior and deployment reliability in modern DevOps processes. Our findings highlight that treating configuration MRs as equivalent to development MRs may obscure meaningful differences. Industrial partners should therefore consider disaggregating review metrics (e.g., time to complete peer review or comment frequency) by MR type to avoid misleading interpretations and to design more precise process interventions.

6.5.2 Implications for researchers

For researchers, the main implication of this chapter is that peer review should not be analyzed as a uniform process across all MRs. The observed differences in review activity, engagement, timeline, and quality across development, configuration, and documentation MRs indicate that aggregate analyses that pool all MR types risk masking category-specific patterns and producing conclusions that do not hold uniformly across artifact types. Future empirical studies on peer review effort or quality should therefore stratify their analyses by MR type, or explicitly report the composition of their MR population, so that their findings can be interpreted and replicated in contexts with different type distributions. The heuristic-based classification used in this chapter (relying on file extensions, folder patterns, and file-type ratios) also provides a reusable operational definition that other studies can adopt to identify configuration and documentation MRs in datasets that lack explicit labels.

The results further suggest that configuration changes deserve a more prominent place in the empirical peer review research agenda. Configuration-dominant MRs show distinct engagement and self-management patterns that are not reducible to those of development MRs, while the amplifying effect of bug presence on configuration-related review intensity and quality highlights the importance of interaction effects between MR type and change context. Studies that examine configuration changes in isolation from development changes, or that ignore the moderating role of bugs, may therefore miss an important part of the picture. Methodologically, this supports the inclusion of MR-type features and bug-presence indicators in regression and predictive models of peer review, and motivates future research on reviewer assignment and review guidelines that are adapted to configuration-specific review needs. Finally, the consistency of the main patterns across industrial and open-source contexts indicates that cross-context replication is feasible when MR-type classification is applied consistently, supporting future comparative studies of peer review across organizational settings.

6.6 Threats to Validity

Internal Validity. The accuracy of our findings depends on the quality and completeness of the data extracted from GitLab. While we ensured that the projects selected are central to the company's configuration activity, this remains a potential limitation. Furthermore, certain factors—such as reviewer experience, domain expertise, or familiarity with SDN—may introduce bias into how reviews are conducted and interpreted.

External Validity. The industrial part of this study is based on MRs collected from TELUS's internal GitLab repositories, which may not fully represent practices in other organizations or SDN deployments. Differences in tooling, team structure, or process maturity may limit the generalizability of our conclusions. To mitigate this, we replicated our analysis on two open-source projects, observing similar patterns and reinforcing the broader applicability of our findings.

Construct Validity. The metrics used to assess reviewer engagement, quality, and self-management may not capture all dimensions of the review process. For instance, factors such as reviewer-author relationship, or the semantic depth of comments are not directly measured. Nevertheless, we rely on established metrics from prior work to ensure comparability and consistency. Our heuristics for classifying MRs (e.g., as configuration-dominant or bug-related) were defined and validated in collaboration with our industrial partner, enhancing their domain relevance. However, these heuristics may not transfer perfectly to all open-source contexts due to differences in project structure or naming conventions. Since we lacked direct feedback from open-source contributors, we manually validated our annotations by closely inspecting MR titles, descriptions, labels, and file changes to ensure consistency and minimize misclassification. Finally, the manual validation of our bug-identification heuristic relied on a random sample of 50 MRs; while this provides a sanity check of the heuristic, the resulting estimate has uncertainty due to sample size (e.g., the worst-case margin of error at $n = 50$ is approximately $\pm 14\%$ when the true proportion is near 0.5).

6.7 Conclusion

In conclusion, this study highlights the challenges and inefficiencies associated with reviewing configuration files, particularly in the context of SDN. Drawing on data from our industrial partner's GitLab MRs, our analyses—both quantitative and qualitative—indicate that configuration-dominant MRs tend to attract less review activity and participation compared to development MRs, yet exhibit higher review quality when bugs are present. Configuration-inclusive MRs, on the other hand, demonstrate greater engagement and higher-quality reviews, mirroring trends observed in development MRs. In comparison to documentation MRs, both configuration-dominant and configuration-inclusive MRs show stronger review participation, higher quality, but more self-managed review process. These findings highlight the need for tailored review practices for configuration files to ensure thorough and effective evaluations. Our insights offer practical guidance for improving review processes and provide valuable tools for practitioners to better manage and assess configuration-related MRs in industrial projects, aligning them more effectively with development and other types of changes.

CHAPTER 7

ARE ALL CODE REVIEWS THE SAME? IDENTIFYING AND ASSESSING THE IMPACT OF MERGE REQUEST DEVIATIONS

Samah Kansab[1] , Francis Bordeleau[1] , Ali Tizghadam[2]

[1] Department of Software Engineering and IT, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[2] TELUS, 25 York St, Toronto, Ontario, Canada M5J 2V5

Paper Published in the 41st IEEE International Conference on Software Maintenance and Evolution on September 2025; extended to the Empirical Software Engineering Journal on May 2026

Preamble

This chapter investigates MRs that do not follow the expected peer review workflow — what we refer to as *deviations* — and their impact on empirical peer review analysis. We identify seven recurrent deviation categories, show that they account for up to 37% of industrial MRs, and demonstrate that few-shot learning can detect them with up to 91% accuracy. Excluding deviations from predictive models of peer review effort substantially changes feature-importance rankings for both time-based and change-based effort measures. Deviations are distributed unevenly across configuration, development, and documentation MRs, and their analytical impact differs across MR types.

This chapter is adapted from a paper published at ICSME 2025 and extended for the Empirical Software Engineering journal (EMSE). The thesis version additionally revises the introduction, conclusion, and overall framing to align the chapter with the objectives of this thesis. Content that overlaps with earlier chapters, such as the MR categorization strategy already presented in Chapter 6, has been removed from this chapter to avoid duplication.

7.1 Introduction

The previous chapters established two foundations for analyzing peer review effort in this thesis. First, Chapter 5 showed that peer review effort should not be measured only through time to complete peer review, and introduced the amount of post-submission change as a complementary,

time-independent indicator of review-induced work. Second, Chapter 6 showed that peer review effort and review quality are not uniform across MR types, since configuration, development, and documentation MRs exhibit distinct review characteristics. Together, these chapters show that effort analysis requires both appropriate metrics and finer-grained categorization of the changes under review.

However, these analyses also rely on an important assumption: that the studied MRs broadly correspond to comparable peer review situations. This assumption does not always hold in practice. In industrial repositories, some MRs are created for purposes such as code cleaning, work in progress, reversions, bulk maintenance tasks, build or configuration adjustments, or temporary coordination. These MRs may bypass, compress, or otherwise depart from the expected review workflow. In this chapter, we refer to such cases as *deviations*, because they depart from the intended peer review process and may distort the interpretation of peer review activity.

This distinction is important because the thesis studies peer review effort through observable MR traces. If deviations are analyzed together with regular peer review cases without distinction, effort-related measures may become difficult to interpret. An MR created for a library update, a build adjustment, a revert, or an incomplete intermediate step may exhibit review patterns, revision activity, and completion dynamics that differ substantially from those of a standard peer-reviewed change. These cases may therefore inflate or suppress time-based and change-based effort measures, alter participation and approval patterns, and bias comparisons across MR categories. From a software process improvement perspective, such bias is problematic because it may lead researchers and practitioners to draw misleading conclusions about where review effort is concentrated and where process optimization is needed.

Deviations may appear similar to outliers or extreme observations, but they represent a different analytical challenge. Outliers are statistical anomalies that differ strongly from typical metric values and are commonly detected through methods such as Local Outlier Factor, Isolation Forest, or One-Class SVM Cheng, Zou & Dong (2019); Chen *et al.* (2016); Mourão-Miranda

et al. (2011). Extremes correspond to the smallest or largest values within a metric distribution and are often identified through threshold- or percentile-based filtering Cais & Pícha (2014); Rajbahadur *et al.* (2019). Deviations, by contrast, are not defined mainly by unusual numerical values, but by departures from the expected workflow structure of peer review. They may involve little recorded review interaction while still being operationally meaningful. Conversely, some regular MRs may also show little visible review activity when discussions occur outside the platform. Purely statistical filtering is therefore insufficient; deviations require contextual reasoning about MR intent, author behavior, and review engagement.

This chapter extends the thesis from measuring and categorizing peer review effort to validating the conditions under which such measurement remains meaningful. It investigates deviations as workflow phenomena that must be identified before effort indicators and MR-type comparisons can be interpreted reliably. More specifically, the chapter pursues five objectives: to characterize common forms of deviations in industrial MR workflows, to examine whether deviation detection can be automated, to analyze how deviations affect predictive modeling based on time to complete peer review, to examine whether the same effect appears when effort is measured through the amount of post-submission change, and to determine whether deviations are distributed differently across MR types.

Across six research questions, we find that workflow deviations are recurrent, accounting for up to 37.02% of MRs in the studied industrial repositories. They organize into seven categories: *Library Update*, *Build or Configuration Adjustments*, *Code Cleaning*, *Revert Commits*, *Experimental or Work in Progress*, *Huge Changes*, and *Empty Change Set*. A few-shot learning approach detects deviations with up to 91% accuracy, making deviation-aware analytics feasible at scale. When deviations are excluded from predictive models of time to complete peer review, model performance is generally preserved or improved, while feature-importance rankings shift substantially. The same pattern extends to predictive models of the amount of changes required to complete peer review, confirming that the bias introduced by deviations is not limited to temporal measures. Moreover, deviations are not uniformly distributed across MR types: configuration MRs are dominated by procedural deviations such as *Library Update* and

Build or Configuration Adjustments, whereas development MRs exhibit a more heterogeneous profile including *Code Cleaning*, *Experimental or Work in Progress*, and *Revert Commits*. Consistently, the analytical effect of excluding deviations differs between configuration and development MRs, especially when effort is measured through post-submission change.

The main contributions of this chapter are:

- a taxonomy of seven workflow-deviation categories in industrial MR repositories, together with an empirical characterization showing that deviations account for a substantial and recurrent share of MR traffic;
- an AI-based detection approach using few-shot learning that reaches up to 91% accuracy, substantially reducing the manual effort required to identify deviation cases at scale;
- evidence that including deviations biases both time-based and change-based predictive models of peer review effort, affecting model interpretation more strongly than raw predictive performance;
- a characterization of how deviations are distributed across configuration, development, and documentation MRs, showing that procedural deviations concentrate in configuration work while development deviations are more heterogeneous;
- evidence that the analytical impact of deviations differs across MR types, indicating that deviation-aware analysis must itself be adapted to the type of artifact under review.

7.2 Study Design

This chapter examines workflow deviations in MRs and their implications for peer review effort analysis. While the previous chapter showed that peer review effort and review quality vary across configuration, development, and documentation MRs, those comparisons still assume that the analyzed cases broadly reflect standard peer review situations. In practice, however, some MRs are created for purposes such as reversions, mechanical maintenance, temporary coordination, or incomplete work, and therefore do not follow the expected review-oriented workflow in the same way as regular cases. As a result, deviations may influence both how peer review effort is measured and how differences across MR types are interpreted. This chapter

therefore investigates deviations not only as workflow phenomena in their own right, but also as a potential source of bias in empirical peer review analysis.

7.2.1 Research Questions

The chapter is guided by the following six research questions:

RQ1 **What are the common types of deviations in the MR review process?** Before deviations can be accounted for analytically, it is necessary to understand what forms they take in practice. MRs may depart from the intended peer review workflow for different reasons, and these reasons are unlikely to be captured adequately through a single generic label. Identifying recurring deviation categories is therefore essential for characterizing how teams use the MR mechanism beyond standard peer review cases and for establishing a foundation for the subsequent analyses of this chapter.

RQ2 **To what extent can AI automate the detection of deviation cases in MRs?** Once deviation categories are identified, manually detecting them at scale becomes impractical, especially in large industrial repositories. Since deviation recognition depends on contextual interpretation rather than on simple threshold-based rules, it is important to examine whether AI-based approaches can support this task reliably. Automating deviation detection is necessary both to reduce manual effort and to enable deviation-aware peer review analytics on large datasets.

RQ3 **What is the impact of excluding MR deviations on the performance and interpretation of models predicting time to complete peer review?** Many studies approximate peer review effort through temporal measures such as time to complete peer review. If deviations systematically differ from regular MRs in their workflow structure, then including them in predictive models may distort both model performance and the interpretation of influential factors. This question therefore examines whether deviation filtering changes conclusions drawn from a commonly used time-based view of peer review effort.

RQ4 What is the impact of excluding MR deviations on the performance and interpretation of models predicting the amount of changes required to complete peer review? The impact of deviations cannot be fully assessed through time-based measures alone. As established in Chapter 5, amount of changes required to complete peer review captures a distinct and more change-oriented dimension of peer review effort. It is therefore necessary to determine whether deviations bias effort analysis only when effort is viewed temporally or whether they also affect effort when it is measured through the amount of changes required to complete peer review.

RQ5 How are MR deviations distributed across configuration, development, and documentation MRs? Although deviations may be common overall, they may not be uniformly distributed across MR types. Configuration, development, and documentation MRs differ in purpose, technical content, and expected review scrutiny. Some deviation categories may therefore be concentrated in specific types of MRs. Examining this distribution is important for avoiding overly general interpretations of review practice and for understanding whether deviations are tied to particular artifact contexts rather than to peer review activity in general.

RQ6 Does the impact of deviations on peer review effort differ across configuration and development MRs? Even if deviations affect peer review effort analysis overall, their impact may not be uniform across MR types. Because configuration and development MRs differ in technical nature and review expectations, the same deviation category may not carry the same meaning or produce the same analytical effect in each context. This question therefore investigates whether deviation-aware analysis should itself be adapted to the type of artifact under review.

7.2.2 Analysis Strategy

The analysis strategy of this chapter is organized at the level of the individual research questions rather than through a single global analytical pipeline. This choice is appropriate because the

chapter addresses several complementary objectives: identifying and characterizing workflow deviations, establishing whether they are empirically distinct from regular MRs, evaluating the feasibility of their automated detection, and examining their consequences for peer review effort analysis. Each objective requires different forms of evidence, distinct preparation steps, and different analytical procedures.

Accordingly, RQ1 focuses on the empirical identification and categorization of deviations in MR workflows. RQ2 then evaluates whether deviation cases can be detected automatically using AI-based classification approaches. RQ3 investigates the impact of deviations on predictive analyses based on *time to complete peer review*, while RQ4 extends this analysis to *amount of changes required to complete peer review*. RQ5 examines how deviations are distributed across configuration, development, and documentation MRs, and RQ6 finally evaluates whether their analytical effect differs depending on the MR type under review.

The study relies on MR traces collected from the five Kaloom technical groups, as described in Section 3.3.1. These traces include information related to MR metadata, changed files, textual descriptions, review interaction, and process outcomes. They are used to characterize deviations, support their detection, compare deviation and regular MRs, and assess the analytical consequences of deviation inclusion or exclusion for peer review effort. Since each research question addresses a distinct aspect of the problem, the corresponding data preparation steps, metrics, and analysis procedures are presented within the approach of each research question rather than through a shared methodological subsection.

Overall, the study design follows a progressive logic. It first establishes a structured understanding of workflow deviations, then examines the feasibility of detecting them automatically, and finally evaluates how their presence affects the measurement and interpretation of peer review effort, both overall and across MR types. This organization supports the broader objective of the thesis by helping ensure that peer review effort is analyzed under workflow conditions that are both empirically grounded and analytically interpretable.

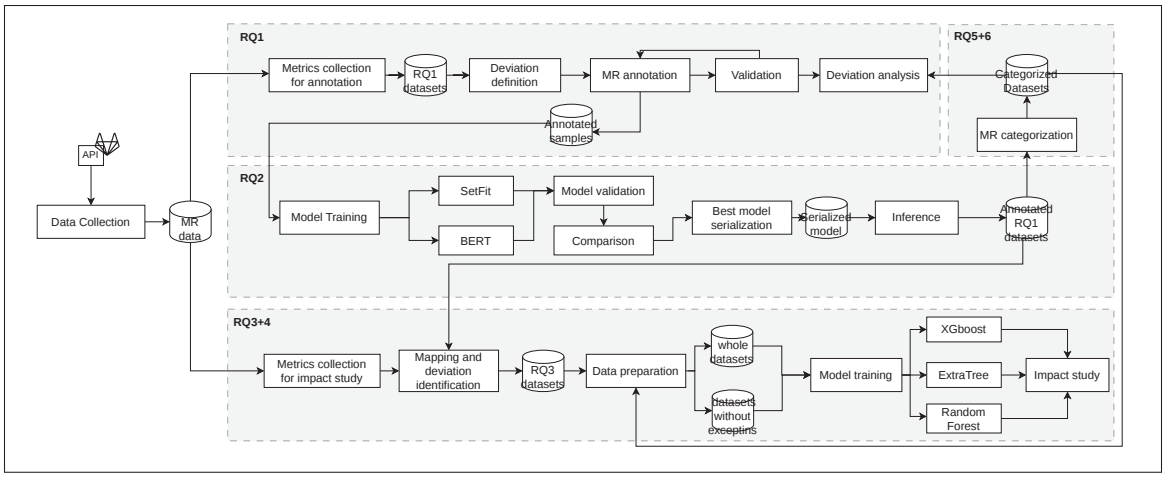


Figure 7.1 Our methodology

7.3 Results

To achieve our goal, we follow a rigorous methodology, as illustrated in Figure 7.1. Using the GitLab API, we use the five Kaloom technical groups (see Section 3.3.1), as summarized in Table 7.3. This dataset enables us to address the following RQs:

7.3.1 Identification and Characterization of MR Deviations

RQ1: What are the common types of MR deviations in the peer review process

Approach

To define and identify MR deviations, we follow these steps:

Metrics Collection

To identify MR deviations, we collect a comprehensive set of metrics designed to capture the entire review activity, from MR creation to closure. These metrics provide insights into various aspects of the process, including time to complete peer review, number of reviewers,

and comment volume—factors that can signal deviations from standard workflows. Table 7.1 presents each metric along with its definition and justification, ensuring our analysis is grounded in meaningful and relevant indicators. However, identifying fixed patterns for each deviation type remains challenging, as many deviations initially resemble normal reviews. For instance, a library update may involve a small change and no reviewer input, yet similar patterns can also occur in legitimate MRs. Keywords-based methods are also insufficient, as they may capture terms unrelated to the intended deviation categories. For example, the keyword "draft" might appear in a title like "[Draft] Initial scaffolding for new feature X", which clearly signals a work-in-progress not intended for review, and thus a deviation. In contrast, a title such as "Refactor module Y – draft ready for review" indicates that the MR, although labelled as a draft, is actually undergoing the standard review process. To address these challenges, we rely on in-depth manual and context-based analysis of the collected metrics, not only regex on metrics, to accurately identify deviations.

Sample Selection

Given the impracticality of manually evaluating thousands of MRs, we adopted a representative sampling approach based on established methodologies Thongtanunam *et al.* (2015). The sample size was calculated using statistical parameters, including the total number of MRs, the desired confidence level, the margin of error, and an estimated population proportion. Initially, the sample size (s) was determined using the formula: $s = \frac{z^2 \cdot p \cdot (1-p)}{c^2}$, where z is the Z-score for the specified confidence level (e.g., 1.96 for 95%), p is the estimated proportion (assumed to be 0.5 for maximum variability), and c is the margin of error (set to 0.05 or 5%). Given the finite MR population, a finite population correction was applied, adjusting the sample size to $ss = \frac{s \cdot P}{P+s-1}$, where P is the total number of MRs. This correction reduces the sample size for smaller populations, ensuring accuracy under finite conditions. The final sample size was rounded up to the nearest whole number to achieve the desired statistical confidence. The sample size of each project is given in Table 7.3.

Table 7.1 Metrics for identifying MR deviations

Metric	Definition	Justification
Title	The title of the MR	Often hints at the MR's purpose.
Description	The description of the MR	Provides more context to identify MR activities.
File types	The list of types of MR files	Specific file types (e.g., '.md' for docs) indicate MR focus.
Code churn	MR total lines of code added+removed	Reflects MR complexity and review effort.
Additions	Lines of code added in the MR	Low or high additions may signal minor fixes or major updates.
Deletions	Lines of code removed in the MR	Large deletions may indicate cleanup or isolated adjustments.
Time to complete peer review	Time from MR creation to merge	Short reviews suggest high priority; long ones indicate low priority.
Reviewer participation	Number of reviewers involved	Few or no reviewers suggest the MR is low-impact or non-essential.
Number of commits	Total commits in the MR	High commits may indicate ongoing work or evolving updates.
Number of committers	Distinct contributors to the MR	Multiple committers may indicate collaborative work or task complexity.
Source branch	Branch from which the MR was created	Branch name may indicate experimental or feature work.
Target branch	Branch into which the MR will be merged	Non-main branches suggest non-critical or work-in-progress changes.
Commit messages	Messages associated with each commit	Can indicate purpose; non-technical tags suggest non-code MRs.
Number of comments	Total comments on the MR	Few comments suggest the MR was not meant for in-depth review.
Comment messages	Content of comments on the MR	Comments content detail the review interaction.
Labels	Tags or labels assigned to the MR	Labels like "documentation" or "build" indicate MR purpose.
Number of reviewers	Number of reviewers of the MR	Multiple reviewers indicate the MR importance
Reviewer comments	Content of reviewer comments	Non-technical comments suggest procedural rather than code-related focus.
Time to first review	Time until the first reviewer engages	Quick engagement suggests high-priority MRs.

Annotation Process

As shown in Figure 7.2, the annotation process for identifying and categorizing MR deviations involves three main phases: definition, annotation, and validation. These phases are iterative and collaborative, ensuring robustness and alignment with industrial insights. The process is carried out by five contributors, including three researchers and two industry partners, combining academic rigor with real-world expertise to ensure the relevance and accuracy of the categorization.

Definition Phase

The process begins by defining the practical use cases of the MR based on real-world scenarios provided by our industrial partners. Concrete MR examples are analyzed to understand how the mechanism is utilized in practice. Identified use cases are then categorized into preliminary deviation types by grouping MRs that diverge from the standard MR process into broader categories. Next, these preliminary categories are refined into formal deviation types through collaborative work sessions involving the entire team. During this phase, categories are further clarified and precisely defined to capture the nuances of each deviation. For each formal category, structured descriptions are provided to distinguish characteristics of these deviations.

Annotation Phase

The annotation phase begins with the manual labelling of randomly selected samples, as described in the previous section. Two researchers collaborate to annotate the samples, ensuring consistency with predefined deviation categories and definitions, while a third researcher supervises the process. Each MR is systematically evaluated to determine its alignment with the identified deviation types, following the formal definitions established in the earlier phases.

Validation Phase

To validate the annotation process, a collaborative inspection phase is conducted in which all annotated MRs are collectively reviewed by the researchers, and random instances are examined by industrial partners. This step addresses ambiguities, refines unclear definitions, and ensures consistency across annotators. Feedback from industrial partners is integrated throughout to align deviation definitions with real-world practices, reinforcing the practical relevance of the identified categories. The evaluators involved in the validation process are senior professionals with over 20 years of experience in software development and peer review, and are also active researchers. The validation demonstrates a high level of agreement, with Cohen's kappa values ranging from 0.78 to 0.95, indicating no major conflicts.

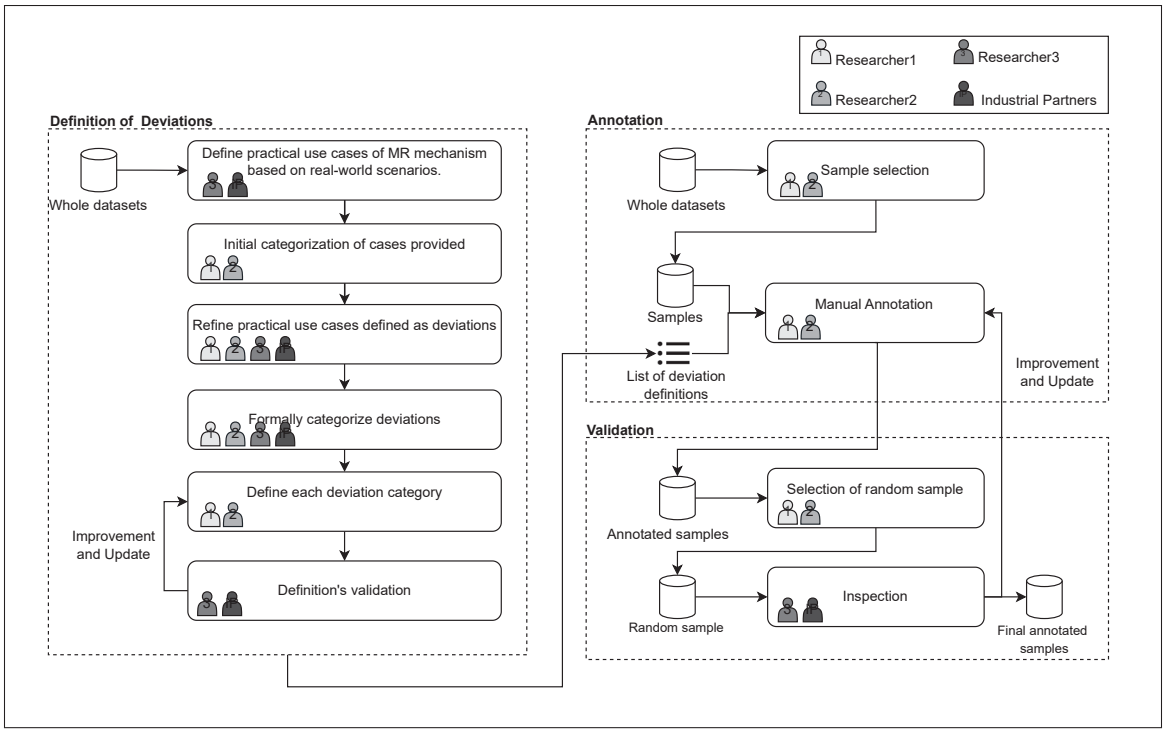


Figure 7.2 The annotation process for definition, annotation and validation of MR deviations

MR Analysis

At this stage, we systematically categorize the occurrence of MR deviations across different projects. This involves analyzing the distribution and frequency of each deviation type, assessing their prevalence within various projects.

Results

As shown in Table 7.3, MR deviations constitute up to 37.02% of cases across seven distinct categories in the analyzed projects (Table 7.2), each representing a deviation from standard review practices. These deviations often indicate scenarios where a traditional, detailed peer review may be deprioritized or streamlined. By systematically categorizing these deviations, Table 7.2 provides a structured classification with clear definitions, enabling quick identification and appropriate handling of MRs that do not necessitate full code scrutiny.

Table 7.2 Deviation types and definitions

Deviation	Definition
Experimental or WIP (Work in Progress) **(EOW)**	MRs created for experimentation or early-stage work, often marked as drafts or WIP. These MRs are not intended for immediate integration and usually do not require thorough review, as they serve as test cases, placeholders, or incomplete implementations.
Code Cleaning **(CC)**	MRs focused on removing outdated, unused, or redundant code without introducing new functionality. These changes do not require extensive review as they do not alter logic or behavior.
Library Update **(LU))**	MRs that solely update the version of a dependency or library without modifying functionality. These changes are typically automated or routine, requiring minimal review as they do not introduce new code.
Build or Configuration Adjustments **(BOCA)**	MRs that modify CI/CD configurations, build scripts, or environment settings (e.g., Docker, Kubernetes) without impacting functional code. These changes are often minor, repetitive, and self-contained, requiring little to no review as they do not affect software behavior or introduce logical errors.
Revert Commits **(RC)**	MRs created to undo previous changes, restoring the codebase to an earlier state. These changes are typically automated and do not require extensive review, as they simply roll back prior modifications without introducing new logic.
Huge Changes **(HC)**	MRs involving large-scale modifications, such as re-branching, code restructuring, or bulk file movements. These changes are often too extensive for a detailed line-by-line review, making traditional review workflows ineffective (More than 500 commits or 10,000+ lines changed.).
Empty Change Set **(ECS)**	MRs that contain no actual modifications, often created unintentionally or due to misconfigured automation. These MRs do not require review, as they introduce no changes to the codebase.

Table 7.3 Descriptive statistics of the studied projects

Project	**Sample size**	**%MR deviations**
Management Plane (MP)	363	22.59
Control Plane (CP)	360	24.44
Data Plane (DP)	359	30.36
FPGA	246	36.59
Platform (PF)	348	37.02

We observe in Figure 7.3 that library updates account for up to 39.3% of all MR deviations, particularly in the CP project, indicating that a significant portion of MRs are created solely for routine dependency adjustments. Similarly, high proportions are observed in the MP project (39. 0%), the DP project (37. 6%) and the PF project (35. 9%). Such updates, typically involving minor version increments without impacting core functionality, do not necessitate a formal review process, as they rarely require reviewers' scrutiny. Explicitly, these MRs include 1-2 addition and 1-2 deletion in 64%-86% of the cases, and no review activity (comments, assignment, or approval) for 94% -96% of the cases. This procedural use of MRs for non-critical changes dilutes the focus of the review process, introducing bias in the related analysis.

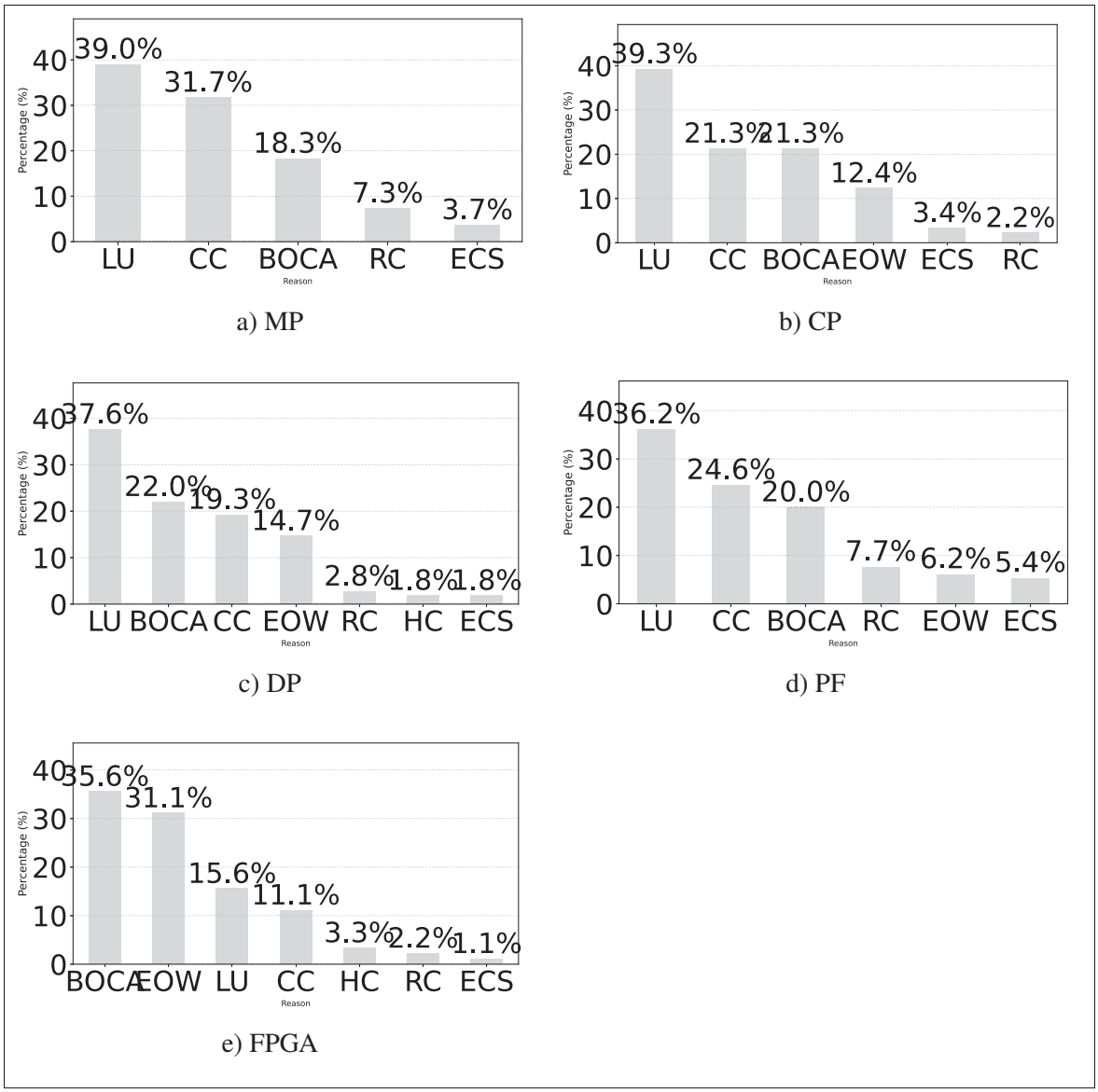


Figure 7.3 Percentage distribution of deviation types across projects (MP, CP, DP, PF, FPGA)

Note: BOCA = Build or Configuration Adjustments, EOW = Experimental or Work in Progress, CC = Code Cleaning, LU = Library Update, RC = Revert Commits, HC = Huge Changes, ECS = Empty Change Set.

The frequency of build or configuration adjustments and code cleaning MR deviations across projects suggests that the MR process is used for changes that do not fundamentally need formal peer review. Build or configuration adjustments deviations are particularly prevalent in the FPGA project, where they account for 35.6% of all MR deviations. Significant proportions are

also observed in the DP project (22.0%), CP project (21.3%), and PF project (19.8%). The high rate of these deviations underscores how the MR process is regularly used for configuration tweaks and CI/CD adjustments, which are typically infrastructure-related and involve minimal impact on the codebase's functionality. These changes, primarily technical setup adjustments, could often bypass the MR process without compromising code quality, freeing up review resources for more critical modifications. This observation is proved by statistics showing that these MRs do not benefit from review activity for 98%-100% of the cases.

Similarly, code cleaning MR deviations—reaching up to 31.7% in the MP project—reflect an emphasis on codebase maintenance through the removal of obsolete code. Other projects also exhibit high percentages, with PF (24.4%), DP (19.3%), and FPGA (15.6%). In these MRs, we observe that the deletions dominant the additions in the churn, as no new code is added. While maintaining a clean codebase is essential, these routine tasks do not require the scrutiny of a formal review while it does not impact the code in use.

Furthermore, experimental or work-in-progress MR deviations, particularly prominent in the FPGA project at 31.1%, represent another instance where MRs are often created without a genuine need for review. These MRs, intended for preliminary testing or drafts, serve primarily as placeholders and are not meant for integration. Similar trends are observed in the CP project (12.4%) and the PF project (6.9%), indicating that MRs are frequently used to manage exploratory changes that could be more efficiently handled in isolated test branches or experimental environments.

In contrast, MR deviations like revert commits, huge changes, quick completion, and empty change sets occur at lower rates (below 7.6% across projects), indicating that these are relatively rare and unlikely to disrupt the review process. The highest occurrence of revert commits deviations is seen in MP (7.3%) and PF (7.7%), while huge changes (HC) appear primarily in FPGA (3.3%) and DP (1.8%). Empty change sets (ECS) remain minimal across projects, with the highest rate at 5.3% in PF.

7.3.2 Automated Detection of MR Deviations

RQ2: To what extent can AI automate the detection of deviation cases in MRs

Approach

To automate MR deviation detection, we compare two AI-based approaches: BERT and SetFit. BERT is known for its high accuracy but requires extensive labeled data and computational resources. In contrast, SetFit enables efficient classification with fewer annotations, making it a suitable choice for scenarios with limited labeled data.

Model Training

BERT transformers

Bidirectional Encoder Representations from Transformers (BERT) is a transformer-based model that learns contextual word representations through self-supervised pretraining on large text corpora Devlin (2018). Following previous studies Yin, Zhao, Sun & Chen (2023); Sharma, Chen, Fard & Lo (2022); Zhang, Cao, Wang & Liu (2023a), we fine-tune BERT for text classification to predict the reason for each MR based on its description.

The dataset is preprocessed by encoding target labels, tokenizing text samples using BertTokenizer, incorporating special tokens, attention masks, and padding for uniform input length. The tokenized data is processed in batches for efficient training. The model is fine-tuned using cross-entropy loss and the AdamW optimizer over multiple epochs.

SetFit

SetFit is a few-shot learning framework designed for text classification tasks where labelled data is limited Tunstall *et al.* (2022). In our experiment, instead of training a traditional classifier, we fine-tune a T5 large language model (LLM) to generate the deviation (or not) for each MR.

Similar to recent works Nguyen *et al.* (2024); Shafikuzzaman (2024), our approach utilizes a small labelled dataset and a few training epochs(5) to minimize computational costs. The model processes input text sequences, learns meaningful representations, and generates predictions in a few-shot setting. This method enables efficient classification while reducing annotation and training overhead, making it practical for large-scale industrial datasets.

Model evaluation

To evaluate model performance, we use commonly adopted classification metrics: Accuracy: Percentage of correct predictions out of total predictions. Recall: Ratio of correctly identified positive cases to actual positive cases. Precision : Proportion of true positive predictions out of all positive predictions. F1-score : Harmonic mean of precision and recall.

Model validation and Comparison

To ensure reliability and account for variability, we perform 10 bootstrap iterations for each model, given the computational cost of training. We report the median results. For each iteration, we split the data into training (80%) and validation (20%) sets. To rank model performance, we use the Scott-Knott Effect Size Difference (ESD) Test[1], a hierarchical clustering method that prioritizes significant performance differences, widely used in prior studies Ouatiti, Sayagh, Kerzazi & Hassan (2022); Rajbahadur *et al.* (2019); Tantithamthavorn & Hassan (2018). The best model is then serialized to be used for the next RQ.

Results

As shown in Table 7.4, our few-shot learning approach with SetFit achieves up to 91% accuracy, outperforming BERT fine-tuning in most cases when sufficient training instances are available. Notably, when using 15 instances per class, SetFit consistently achieves the highest performance across all projects. BERT fine-tuned for 5 epochs demonstrates strong performance in certain

[1] https://cran.r-project.org/web/packages/ScottKnottESD/

cases, achieving an accuracy up tp 0.84, but still does not surpass SetFit's top results. When BERT is trained for only 3 epochs, its accuracy drops significantly, down to 0.53 showing the lowest results on PF dataset, indicating that larger fine-tuning is necessary even using all the datasets for better performance.

In addition to accuracy, we observe that SetFit at 15 instances per class achieves an average F1-score of 0.84 across projects, whereas BERT-5 reaches 0.74. The precision scores for SetFit (0.75 on average) also surpass those of BERT (0.65). More notably, recall values for SetFit (0.91 at 15 instances) demonstrate superior generalization compared to BERT (0.76 at 5 epochs).

Overall, SetFit with 15 instances per class and the same number of epochs as BERT ranks highest, showing its ability to generalize well with a small labeled dataset compared to BERT. The performance gap between SetFit and BERT increases as the number of training instances grows, reaffirming SetFit's robustness in few-shot learning scenarios.

Table 7.4 The median performances of each algorithm

Project	Instances	SetFit (5 Epochs)			
		Accuracy	Precision	Recall	F1-score
PF	5	0.600	0.550	0.650	0.600
DP		0.590	0.520	0.620	0.570
FPGA		0.500	0.450	0.550	0.500
MP		0.420	0.400	0.500	0.440
CP		0.500	0.450	0.550	0.500
PF	10	0.800	0.750	0.850	0.800
DP		0.770	0.700	0.800	0.750
FPGA		0.769	0.720	0.820	0.770
MP		0.751	0.680	0.780	0.730
CP		0.700	0.650	0.750	0.700
PF	15	**0.910**	0.780	0.980	0.870
DP		0.870	0.740	0.940	0.830
FPGA		0.800	0.780	0.980	0.870
MP		0.790	0.710	0.910	0.800
CP		0.900	0.730	0.930	0.820
Project	Epochs	BERT			
		Accuracy	Precision	Recall	F1-score
PF	3	0.530	0.280	0.530	0.360
DP		0.710	0.600	0.710	0.610
FPGA		0.660	0.435	0.660	0.520
MP		0.670	0.450	0.670	0.540
CP		0.640	0.410	0.640	0.500
PF	5	0.810	0.780	0.810	0.800
DP		0.840	0.820	0.840	0.800
FPGA		0.680	0.610	0.680	0.630
MP		0.790	0.710	0.790	0.740
CP		0.760	0.670	0.760	0.700

7.3.3 Impact of MR Deviations on Peer Review Effort Measures

7.3.3.1 Time to Complete Peer Review

RQ3: What is the Impact of Excluding MR Deviations on Time to Complete Peer Review Model Performance and Interpretation

Approach

To assess the impact of removing deviations on the models predicting time to complete peer review, we compare models trained on the full set of MRs with those trained on a subset excluding deviations. The comparison follows these steps:

Data Preparation

Data Annotation

Using the best-performing model from RQ2—SetFit with 15 instances per class—we classify the entire dataset to determine whether each MR exhibits deviation. To ensure reliability, we randomly sample and validate predictions with our industrial partners of the new predictions.

Metrics Collection

While the initial metrics were designed to predict deviations, this step focuses on extracting features that explain the time to complete peer review. The selected metrics, derived from prior studies (Chouchen *et al.*, 2023; Hasan *et al.*, 2023; McIntosh *et al.*, 2016a; Thongtanunam *et al.*, 2017b). We ensured that all the metrics can be collected at the moment of creation of the MR. The metrics, with their description, are summarized in Table 7.5.

Table 7.5 Metrics for time to complete peer review prediction

Metric	Description	Metric	Description
Delta time	Time between project creation and MR creation	Associated sprint	The sprint ID in which the MR was created
#Committers	Number of MR committers	Has_Assignee	Indicates whether the MR has an assigned reviewer
#Authored_Commits	Number of commits authored before MR creation	#Committed_commits	Number of committed changes
Change churn	Total number of lines added and removed in the MR	Additions	Number of lines added
Deletions	Number of lines deleted	Title length	Length of the MR title
Description length	Length of the MR description	Is_Hashtag	Indicates whether the MR description contains a hashtag
Is_AtTag	Indicates whether the MR description contains an @mention	Source branch MRs	Number of open MRs on the same source branch
Source branch approval time	Average approval time for source-branch MRs	Target branch MRs	Number of open MRs targeting the same branch
Target branch approval time	Average approval time for target-branch MRs		
#Minor contributors	Number of MR contributors with less than 5% of prior changes	#Major contributors	Number of contributors with more than 5% of prior changes
Self-approved MRs	Number of historically self-approved MRs	MRs without discussion	Number of historical MRs with no discussion
Avg. historical approval time	Average approval time for historical MRs	#Open MRs_history	Number of historical open MRs
Historical MR size	Average size of past MRs	Historical entropy	Measure of historical change diversity
#Initial files	Number of files		

Collinearity Analysis

To enhance model reliability and interpretability, we mitigate collinearity through correlation and redundancy analysis, following Tantithamthavorn *et al.* (2016b) and Cito *et al.* (2021), as well as previous studies (Jiarpakdee *et al.*, 2019; Tantithamthavorn & Hassan, 2018). First, we apply Spearman correlation analysis to remove highly correlated features, defining two metrics as correlated if their coefficient exceeds 0.70, as suggested by McIntosh *et al.* (2014) and Lee *et al.* (2020). Since correlation alone does not fully eliminate collinearity, we also perform redundancy analysis using the R^2 value to identify independent features that can be explained by others. A threshold of 0.90 is applied, following prior work (Tantithamthavorn *et al.*, 2018; Lee *et al.*, 2020), to filter out redundant features.

Model Training and Evaluation

Following Chouchen *et al.* (2023), we predict time to complete peer review, a key metric balancing prediction and interpretability. We employ ExtraTrees, XGBoost, and Random Forest, identified as top-performing algorithms in prior studies. Model performance is evaluated using SA (Standard Accuracy), MAE (Mean Absolute Error), and MSE (Mean Squared Error). For statistical robustness, we apply a 100 out-of-sample bootstrap technique (Rajbahadur *et al.*, 2017; Tantithamthavorn *et al.*, 2016b) to ensure stable conclusions. We use feature importance calculation for each of the 100 bootstrap samples.

Impact Assessment

Performance

To assess the impact of removing MR deviations, we compute the ratio of the median bootstrap performance of model without deviations to that with deviations (Rajbahadur *et al.*, 2019). If lower values indicate better performance, we use 1 − median. For instance, if the median MSE without deviations is 0.2 and with deviations is 0.5, the improvement factor is $(1 - 0.2)/(1 - 0.5) = 1.6$. Conversely, if SA improves from 0.4 to 0.8, the improvement is $0.8/0.4 = 2$ times. For significance testing, we apply Cliff's Delta and Wilcoxon tests between the 100 bootstrap results for each model.

Interpretation

For each model (those with and those without deviations), we use feature importance calculation for each of the 100 bootstrap samples, we then compute rankings per bootstrap and aggregate them using Scott-Knott ESD to derive a single rank per model. To assess rank similarity, we apply Kendall's Tau (τ), previously used in (Rajbahadur, Wang, Oliva, Kamei & Hassan, 2021; Bachmann & Bernstein, 2010; Akoglu, 2018). This method uses Spearman's rank correlation to compare the similarity between two feature importance ranks. A Tau correlation of 1 indicates total similarity, while -1 indicates total dissimilarity. For the top-5 features, we use the Top-K overlap method (Jaccard, 1901), as used by Rajbahadur *et al.* (2021), measuring intersection divided by union for top-1, top-3, and top-5 features across iterations.

Results

Removing deviations results in significant improvements in 53.33% of cases, with a large effect size and a p-value<0.05, indicating statistical significance. In 24.24% of cases, no measurable impact is observed. Regarding the XGBoost algorithm, we observe improvements in 60% of the cases across all models, except for the PF project, where MAE and SA significantly decrease by

Table 7.6 Performance and feature importance metrics for different models

Model	XGBoost							ExtraTree							Random Forest						
	Performance			Feature Importance				Performance			Feature Importance				Performance			Feature Importance			
	MSE	MAE	SA	K	T1	T3	T5	MSE	MAE	SA	K	T1	T3	T5	MSE	MAE	SA	K	T1	T3	T5
PF	0.99	*0.99+*	*0.96++*	*0.74++*	*1++*	*1++*	0.42*	**1.02**	**1**	1.01	0.78++	1++	1++	0.42*	0.99	0.99	1	0.81++	1++	1++	0.42*
DP	***1.02***	***1.01***	*1++*	0.35+	1++	0.2	0.25	1	1	*0.98++*	0.49+	1++	0.2	0.25	*1+*	***1***	***1***	0.66++	1++	0.5*	0.42*
FPGA	1.01	1.01	***0.98***	0.67++	1++	0.50+	0.42*	***1.01***	1	0.92	0.54+	1++	0.50*	0.42*	0.99	0.99	0.84	0.67++	1++	0.50*	0.42*
MP	***1.02***	***1.03***	***1.07***	0.60+	1++	0.50+	0.25	*1.01++*	0.99	0.99	0.55+	1++	0.20	0.11	***1.02***	***1.00***	*0.99++*	0.71++	1++	0.20	0.42*
CP	***1.10***	***1.03***	***2.25***	0.16*	0	0.20	0.11	***1.08***	***1.02***	1	0.50+	0	0.20	0.42*	***1.11***	***1.02***	***1.09***	0.66++	0	0.20	0.42*

Cohen's d: Negligible if $|d| \leq 0.147$, Small+ if $0.147 < |d| \leq 0.33$, Medium++ if $0.33 < |d| \leq 0.474$, **Large** if $|d| > 0.474$.
Wilcoxon Test: *Significant* if $p - value \leq 0.05$
Kendall's Tau (k): Weak* if $|\tau| \leq 0.3$, Moderate+ if $0.3 < |\tau| \leq 0.6$, Strong++ if $|\tau| > 0.6$.
Top-K (Tk: T1, T2, T3) Overlap: Negligible if $0.0 < Tk \leq 0.25$, Small* if $0.25 < Tk \leq 0.50$, Medium+ if $0.50 < Tk \leq 0.75$, Large++ if $0.75 < Tk \leq 1$.

factors of 0.99 and 0.96, respectively. This improvement is most pronounced in the CP project, where SA performance increases up to 2.25 times, rising from 0.16 to 0.25. For ExtraTree and Random Forest, we observe significant improvements or no negative impact in 80% and 60% of cases, respectively, while in 20% and 40% of cases, the decrease remains statistically insignificant in 94% of the cases for both algorithms. For instance, in the MP project with random forest, MSE improves by a factor of 1.02, decreasing from 0.22 to 0.20.

Kendall's tau results indicate weak similarity in 6.67% of cases and moderate similarity in 40% of cases, highlighting the substantial impact of removing deviations on feature importance rankings. As shown in Table 7.6, removing deviations impacts the interpretability of XGBoost in three out of five projects, with the CP project experiencing the most substantial dissimilarity, dropping to 0.16. Similarly, for ExtraTree, we observe a moderate similarity in four out of five projects. In contrast, the feature importance rankings in Random Forest are less impacted, as all Kendall's tau values are strong, indicating high similarity. However, since none of these values equal one, some features still experience minor shifts in their importance.

Regarding Top-k features commonly used in the literature, we observe weak overlap in 26.67% of cases and small overlap in 33.33% of cases, underscoring the necessity of considering deviation removal when analyzing the peer review process. More specifically, **for Top-1** features, we observe that the most important variable changes for all algorithms in the CP project. For instance, in the CP project with ExtraTree, the most critical variable shifts from the number

of files to the number of commits, which alters its interpretability. The number of files could indicate the extent of modification, whereas the number of commits reflects the frequency of contributions. This change can impact how review complexity is assessed.

For **Top-3** features, we observe negligible overlap in 46.67% of cases and small overlap in 33.33%, indicating a substantial impact. A concrete example, in the DP project with XGBoost, the top three features shift from *title length, description length, associated sprint* to *number of commits, associated sprint, number of historical open MRs*, demonstrating that deviation removal significantly influences feature selection. Similarly, **for Top-5** features, we observe negligible overlap in 33.33% of cases and small overlap in 66.67%, further emphasizing the extensive impact across all case studies. These results reinforce the importance of considering deviations when interpreting the peer review process.

7.3.3.2 Amount of of changes required to complete peer review

RQ4: What is the impact of excluding MR deviations on the performance and interpretation of models predicting the amount of changes required to complete peer review

Approach

In addition to *time to complete peer review*, we examine a second and complementary dimension of peer review effort, namely the *amount of changes required to complete peer review*. Following Chapter 5, this measure captures the amount of modification performed after MR creation and before MR completion, and is calculated as the total number of lines added and deleted during this post-submission period. Unlike time to complete peer review, which reflects the temporal progression of the review process, this metric captures how much the submitted change had to evolve before it became acceptable. It therefore provides a more direct view of the change effort associated with MR completion.

To assess the impact of deviations on this second effort dimension, we followed the same analytical procedure used for the previous RQ. More specifically, for each project, we compared models trained on the full set of MRs with models trained on datasets from which deviations had been excluded. We then evaluated the impact of deviation exclusion on both predictive performance and model interpretation. The same learning algorithms, validation procedure, and interpretation measures were used as in the previous RQ, allowing direct comparison between the two effort dimensions.

Results

Table 7.7 shows that deviations also affect effort analysis when effort is measured through the amount of changes required to complete peer review. Overall, the results reveal a pattern consistent with that observed for *time to complete peer review*: excluding deviations generally improves, or at least preserves, predictive performance, while substantially altering the interpretation of the resulting models.

Excluding deviations leads to statistically significant performance improvements in 46.67% of cases, while 33.33% of cases show no measurable impact and only 20.00% exhibit significant degradation. Across the 15 project–model combinations, the improvements are concentrated primarily in MSE and MAE. For XGBoost, significant improvements are observed in 60.00% of cases, covering PF, DP, MP, and CP. The strongest gain appears in DP, where MSE improves by a factor of 2.96 and MAE by 1.57. ExtraTree shows the strongest overall benefit, with significant improvements in 66.67% of cases, again led by DP, where MSE improves by 3.15 and MAE by 1.91. Random Forest presents a more mixed but still generally positive pattern, with significant improvements in 40.00% of cases, notably in DP, MP, and CP. These results indicate that deviations do not affect only the temporal dimension of peer review effort, but also the amount of change required after submission before MR completion.

When considering predictive performance more broadly, excluding deviations improves or preserves model performance in 88.89% of the evaluated cases. In several cases where

improvements are not statistically significant, the performance ratios remain close to 1, suggesting that removing deviations does not harm predictive accuracy. This stability is particularly visible for Random Forest, where several results remain close to parity even when the gains are smaller than those observed for XGBoost and ExtraTree. Only a limited number of cases show significant degradation, and these are concentrated mainly in SA. For example, PF exhibits a decrease in SA for ExtraTree and Random Forest, and FPGA shows a smaller negative impact for ExtraTree. Nevertheless, the dominant trend remains positive: removing deviations generally improves the ability of ML models to predict the amount of changes required to complete peer review, or at minimum leaves it unaffected.

The interpretation results reveal an even stronger impact of deviations. Kendall's Tau indicates that feature importance rankings are consistently altered, with 93.33% of cases showing strong shifts and the remaining 6.67% showing moderate shifts. More specifically, all but one project–model combination yield Tau values above 0.60, demonstrating that the explanatory structure of the models changes substantially once deviations are excluded. This impact is especially pronounced for ExtraTree and Random Forest, where several projects reach values above 0.85, such as MP, CP, and FPGA. These results show that even when predictive performance changes are modest, the variables emphasized by the models can differ markedly between the full and deviation-filtered datasets.

The Top-k analysis further confirms that excluding deviations substantially changes which variables are identified as the most important predictors. For Top-1, we observe negligible overlap in 93.33% of cases and complete overlap in only one case (6.67%), namely Random Forest in FPGA. This means that in nearly all settings, the single most important predictor changes once deviations are excluded. For Top-3, 40.00% of cases show negligible overlap and 60.00% show small overlap, with no case reaching medium or large agreement. For Top-5, 46.67% of cases show small overlap and 53.33% show medium overlap, again indicating that the feature sets highlighted by the models remain only partially consistent across the two settings. Thus, deviation exclusion does not merely produce minor reordering among predictors; it

frequently changes which factors appear among the most influential drivers of the amount of changes required to complete peer review.

Taken together, these findings confirm that the analytical impact of deviations extends beyond *time to complete peer review*. Even when effort is measured through the amount of changes required to complete peer review, deviations still affect both predictive performance and model interpretation. This reinforces the broader argument of this RQ that deviations are not harmless irregularities in MR analytics. Instead, they constitute a meaningful source of bias that can distort multiple dimensions of peer review effort and lead to misleading conclusions when they are ignored.

Overall, the results show that excluding deviations generally improves, or at least does not harm, the prediction of the amount of changes required to complete peer review, while substantially modifying feature importance rankings. This confirms that the impact of deviations is not limited to time to complete peer review, but also extends to a second complementary dimension of peer review effort.

Table 7.7 Performance and feature importance metrics for different models (MR size prediction)

Model	XGBoost							ExtraTree							Random Forest						
	Performance			Feature Importance				Performance			Feature Importance				Performance			Feature Importance			
	MSE	MAE	SA	K	T1	T3	T5	MSE	MAE	SA	K	T1	T3	T5	MSE	MAE	SA	K	T1	T3	T5
PF	0.98	***1.18****	***1.15**++*	0.64++	0	0.50*	0.43*	**1.26**	***1.36**++*	*0.90++*	0.66++	0	0.50*	0.43*	1.00	*1.00++*	*0.52++*	0.68++	0	0.20	0.43*
DP	***2.96**++*	***1.57**++*	***1.74**++*	0.68++	0	0.20	0.25	***3.15**++*	***1.91**++*	*0.95++*	0.75++	0	0.50*	0.67+	*1.00++*	*1.00++*	*0.67++*	0.80++	0	0.50*	0.43*
FPGA	***1.09***	***1.11***	**1.02**	0.75++	0	0.50*	0.43*	**1.01**	**1.09**	*0.94**	0.80++	0	0.50*	0.43*	1.00	*1.00**	*0.68++*	0.92++	1.00++	0.50*	0.67+
MP	***2.28**++*	***1.64**++*	***1.04****	0.41+	0	0.20	0.43*	***2.29**++*	***1.78**++*	*0.94++*	0.87++	0	0.50*	0.67+	*1.00++*	*1.00++*	*0.57++*	0.87++	0	0.50*	0.67+
CP	***1.89**++*	***1.49**++*	1.00	0.71++	0	0.50*	0.67+	***2.00**++*	***1.66**++*	*0.98+*	0.94++	0	0.50*	0.67+	*1.00++*	*1.00++*	*0.65++*	0.91++	0	0.50*	0.67+

Cohen's d: Negligible if $|d| \leq 0.147$, Small+ if $0.147 < |d| \leq 0.33$, Medium++ if $0.33 < |d| \leq 0.474$, **Large** if $|d| > 0.474$.
Wilcoxon Test: *Significant* if $p - value \leq 0.05$
Kendall's Tau (k): Weak* if $|\tau| \leq 0.3$, Moderate+ if $0.3 < |\tau| \leq 0.6$, Strong++ if $|\tau| > 0.6$.
Top-K (Tk: T1, T2, T3) Overlap: Negligible if $0.0 < Tk \leq 0.25$, Small* if $0.25 < Tk \leq 0.50$, Medium+ if $0.50 < Tk \leq 0.75$, Large++ if $0.75 < Tk \leq 1$.

7.3.4 Distribution of MR Deviations Across MR Types

RQ5: How are MR deviations distributed across configuration, development, and documentation MRs

Approach

To examine whether deviations vary according to the nature of the changed artifacts, we classified each MR into one of three categories: *configuration*, *development*, or *documentation*. This classification follows the heuristic-based strategy proposed in our earlier work on MR type analysis in Chapter 6, while being adapted to the file naming conventions and project structure of our industrial context.

Using the deviation labels obtained in the previous RQ, we then analyzed the distribution of deviation types within each MR category across all projects. We first report the overall distribution of MR types per project. We then examine how the identified deviation categories are distributed within configuration, development, and documentation MRs.

Results

Figure 7.4 presents the overall distribution of MR types across the five projects. **We observe that MRs are predominantly development-oriented in most projects. The only exception is FPGA, where configuration MRs slightly outnumber development MRs, representing 54.1% and 45.5% of MRs, respectively**. This result is consistent with the nature of FPGA-related work, which relies heavily on hardware-oriented setup, configuration artifacts, and infrastructure-sensitive adjustments. In contrast, documentation MRs are almost absent across all projects. They are entirely absent in PF and CP, and remain below 1% in MP, DP, and FPGA, with proportions of 0.3%, 0.8%, and 0.8%, respectively. Because of this very low frequency, our analysis mainly focuses on configuration and development MRs.

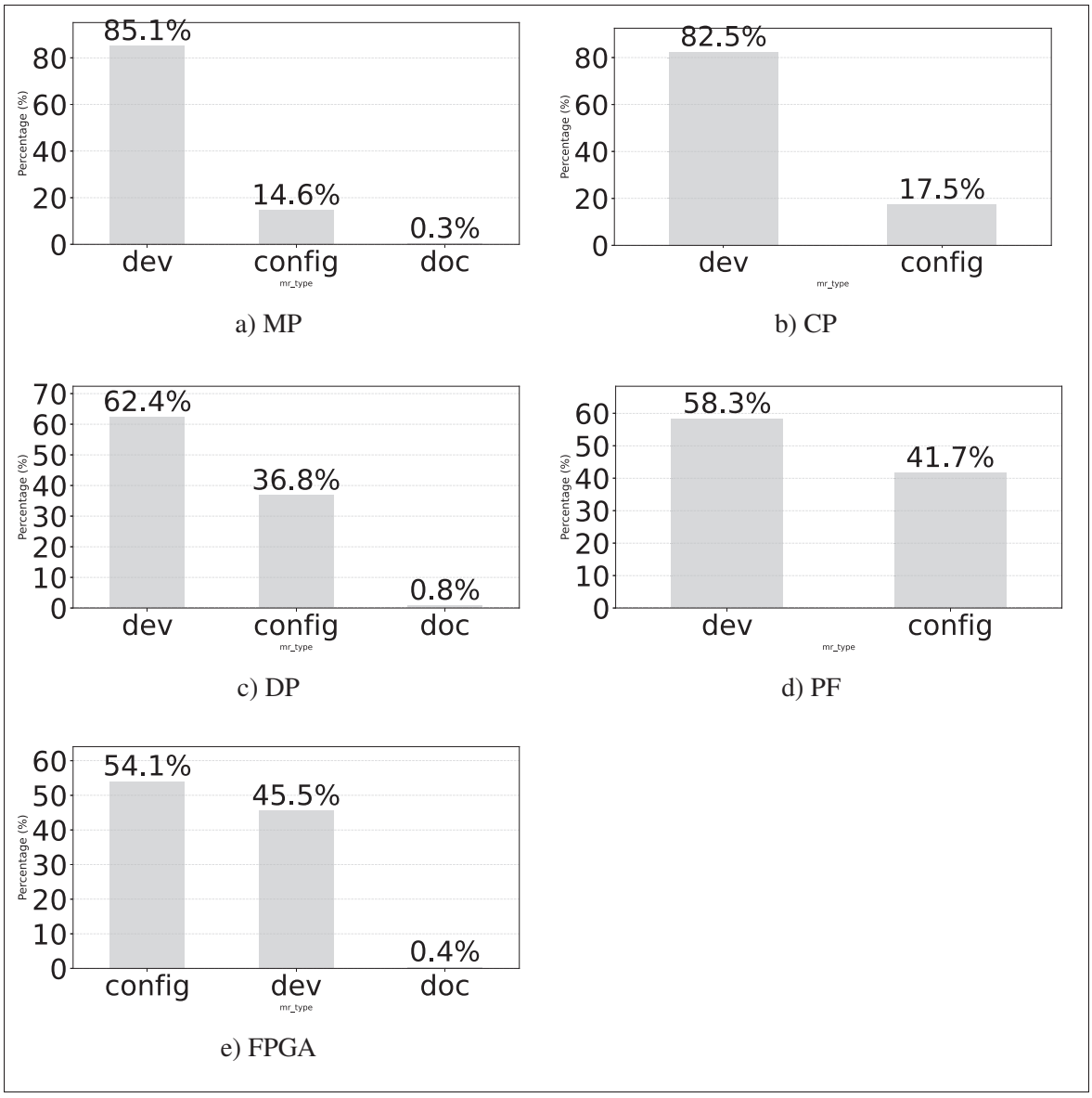


Figure 7.4 Percentage distribution of MR types across projects (MP, CP, DP, PF, FPGA)

Figure 7.5 shows that the distribution of deviation types is not uniform across MR types. Instead, deviations exhibit clear specialization patterns depending on whether the MR is related to configuration, development, or documentation. Across most projects, configuration MRs are dominated by *Library Update* (LU) and *Build or Configuration Adjustments* (BOCA), confirming that deviations in this MR type are largely associated with operational maintenance

and infrastructure-related work rather than with substantial review needs. For example, in the PF project, configuration deviations are mainly composed of LU (62.1%), followed by BOCA (13.8%) and *Code Cleaning* (CC) (12.1%), while development deviations in the same project are more dispersed, with CC (38.2%) becoming the most frequent type, ahead of BOCA (26.5%) and LU (17.6%). In the MP project, a similar pattern is observed, where configuration deviations are again largely driven by LU (55.2%), whereas development deviations are more balanced across CC (33.3%), BOCA (21.2%), and LU (21.2%). Likewise, in the CP project, configuration deviations are mostly LU (50.0%) and BOCA (27.3%), while development deviations are spread across BOCA (32.9%), CC (23.3%), and LU (21.9%).

A similar but even more pronounced contrast appears in the DP and FPGA projects. In DP, configuration deviations are dominated by LU (54.2%) and BOCA (25.4%), whereas development deviations shift toward Experimental or Work in Progress (EOW) (32.7%), followed by CC (25.5%) and LU (21.8%). Documentation deviations are rare, but when present, they are split equally between LU and BOCA. In FPGA, the distribution departs from the previous projects by showing that configuration deviations are primarily BOCA (55.3%), followed by LU (21.3%) and EOW (14.9%). This result is coherent with the hardware-oriented and infrastructure-sensitive nature of the project. Development deviations in FPGA are more heterogeneous, with BOCA (34.5%), CC (27.6%), and *Revert Commits* (RC) (20.7%) all contributing substantial shares. Documentation deviations in FPGA are exclusively RC (100%), although this likely reflects a very small number of documentation-related deviation cases.

Taken together, these results show that the same MR type does not always imply the same level of review relevance. In particular, even when configuration MRs appear important because they concern configuration artifacts, a substantial proportion of them correspond to deviations such as *Library Updates* and *Build or Configuration Adjustments*. These cases often reflect routine version changes, environment updates, or technical setup modifications that may appear operationally important, yet do not necessarily require substantial peer review effort. As a result, configuration MRs can create the impression of demanding careful review while, in practice,

many of them correspond to lightweight or procedural changes that bypass the need for in-depth evaluation.

A related but distinct phenomenon appears in development MRs. Although development changes are generally assumed to represent the core of peer-reviewed software work, the deviation profile within this MR type shows that a notable portion of development MRs also do not fully reflect standard review scenarios. Instead of being dominated by routine configuration-oriented cases, development deviations are more heterogeneous and include *Code Cleaning*, *Experimental or Work in Progress*, and, in some projects, *Revert Commits*. This suggests that some development MRs that may initially appear to deserve regular review attention are in fact associated with cleanup activities, exploratory work, or rollback operations that do not carry the same review objectives as regular implementation changes.

From an analytical perspective, these findings are important because they show that the apparent nature of an MR is not always sufficient to infer the level or type of review effort it actually requires. A configuration MR is not necessarily a high-attention review case, just as a development MR is not necessarily a standard implementation review. Consequently, treating deviations as a single homogeneous phenomenon, or assuming that MR type alone adequately characterizes review needs, can mask important differences in how the MR mechanism is used in practice. This reinforces the need for deviation-aware analyses that account not only for whether an MR is a deviation, but also for the type of artifact involved when interpreting review practices or building predictive models of peer review effort.

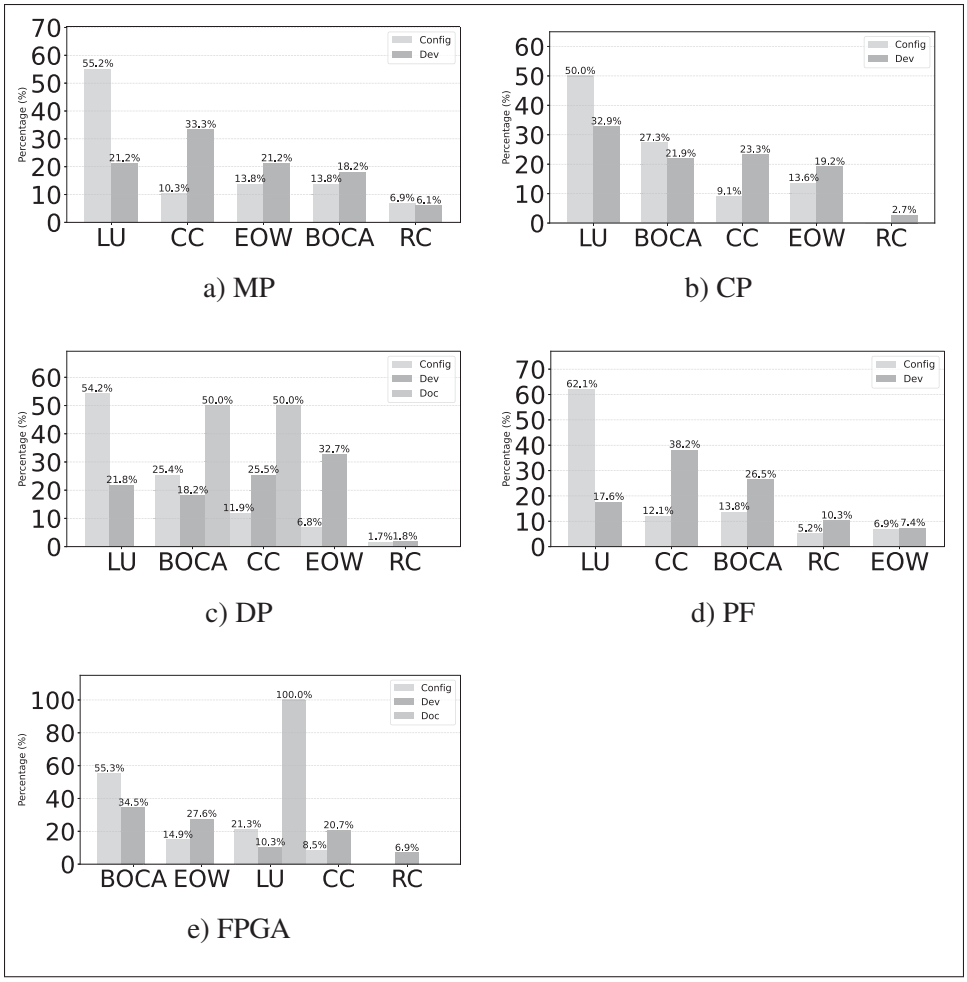


Figure 7.5 Percentage distribution of deviation categories across projects (MP, CP, DP, PF, and FPGA) by MR type

Note: BOCA = Build or Configuration Adjustments, EOW = Experimental or Work in Progress, CC = Code Cleaning, LU = Library Update, RC = Revert Commits, HC = Huge Changes, ECS = Empty Change Set.

7.3.5 Differential impact of MR Deviations Across MR Types

RQ6. Does the impact of deviations on peer review effort differ across configuration and development MRs

Approach

After examining the overall impact of deviations on peer review effort and showing that deviations are distributed differently across MR types, we next investigate whether their analytical impact

also varies by MR type. This RQ extends the previous analyses by moving from a global view of all MRs to a type-specific analysis of configuration and development MRs.

To address this question, we follow the same analytical principle used in the previous effort-related RQs, but we perform the analyses separately for each MR type. More specifically, we partition the dataset into *configuration MRs* and *development MRs*, and for each type we compare models trained on the full set of MRs with models trained after excluding deviations. As in the earlier analyses, we evaluate the impact of deviation exclusion on both predictive performance and model interpretation. We consider the same two peer review effort measures used previously: *time to complete peer review* and *amount of changes required to complete peer review*. Documentation MRs are excluded from this analysis because they are too rare to support stable and meaningful model construction.

Tables 7.8, 7.9, 7.10, and 7.11 summarize the results obtained for configuration and development MRs across the studied projects.

Results

The results show that the impact of excluding deviations differs across MR types and across effort measures. When effort is measured through *time to complete peer review*, the impact of deviation exclusion remains generally limited for both configuration and development MRs, although it is overall less favorable for configuration MRs. For configuration MRs, excluding deviations leads to significant performance improvements in only 6.67% of cases, while 55.56% of cases show no measurable impact and 37.78% exhibit significant degradation. These degradations are concentrated almost entirely in SA, suggesting that removing deviations does not improve the temporal predictability of configuration MRs and may, in some settings, reduce it. In contrast, development MRs show a more balanced pattern: 22.22% of cases improve significantly, 60.00% show no measurable impact, and 17.78% degrade significantly. Thus, while the time-based impact of deviations remains modest overall, it is more favorable in development MRs than in configuration MRs.

The contrast becomes much stronger when effort is measured through amount of changes required to complete peer review. For configuration MRs, excluding deviations produces significant performance improvements in 53.33% of cases, while 22.22% of cases show no measurable impact and 24.44% exhibit significant degradation. These improvements are especially pronounced for XGBoost and Random Forest, with very large gains in projects such as DP and MP. For development MRs, the pattern is also positive but slightly more mixed: 51.11% of cases improve significantly, 17.78% show no measurable impact, and 31.11% degrade significantly. Across both MR types, the dominant impact is therefore much stronger for amount of changes required to complete peer review than for time to complete peer review, indicating that deviations have a clearer and more systematic impact on change-based effort than on temporal effort.

When considering performance more broadly, excluding deviations preserves or improves results in 62.22% of cases for configuration MRs and 82.22% of cases for development MRs when predicting time to complete peer review. For amount of changes required to complete peer review, the corresponding proportions rise to 75.56% for configuration MRs and 68.89% for development MRs. These results indicate that deviations do not impact all MR types in the same way. In the temporal dimension, development MRs appear more robust to deviation filtering, whereas in the change-based dimension, both MR types benefit substantially, with configuration MRs showing the clearest gains in several projects.

The interpretation results also reveal type-dependent impacts. For configuration MRs, Kendall's Tau values for *time to complete peer review* indicate weak similarity in 6.67% of cases, moderate similarity in 33.33%, and strong similarity in 60.00%. For development MRs, the impact is slightly stronger, with 33.33% of cases showing moderate similarity and 66.67% showing strong similarity. When considering *amount of changes required to complete peer review*, the interpretation shift becomes even clearer. For configuration MRs, 6.67% of cases show weak similarity, 26.67% moderate similarity, and 66.67% strong similarity. For development MRs, 40.00% of cases show moderate similarity and 60.00% strong similarity. Overall, these results

show that even when predictive performance remains relatively stable, the explanatory structure of the models often changes after removing deviations, and this occurs across both MR types.

The Top-k overlap analysis further confirms that the impact of deviations is not uniform across MR types or effort measures. For *time to complete peer review*, the Top-1 feature remains identical before and after deviation exclusion in 100% of cases for both configuration and development MRs, suggesting that the single most important predictor remains stable in the temporal setting. However, this apparent stability weakens when broader feature sets are considered, as Top-3 and Top-5 overlaps remain only partial in many cases. For *amount of changes required to complete peer review*, the pattern is markedly different. In configuration MRs, Top-1 overlap is negligible in 53.33% of cases and complete in 46.67%, whereas in development MRs it is negligible in 66.67% of cases and complete in only 33.33%. This indicates that the most important predictor is much more likely to change when deviations are removed from the amount of changes required to complete peer review models, especially for development MRs. Thus, the interpretation impact of deviations is not only measure-dependent, but also MR-type dependent.

Table 7.8 Performance and feature importance metrics for configuration MRs (time to complete peer review prediction)

Model	XGBoost							ExtraTree							Random Forest						
	Performance			Feature Importance				Performance			Feature Importance				Performance			Feature Importance			
	MSE	MAE	SA	K	T1	T3	T5	MSE	MAE	SA	K	T1	T3	T5	MSE	MAE	SA	K	T1	T3	T5
PF	1.00	*1.00*	*0.96**	0.39+	1.00++	0.50*	0.25	1.00	1.00*	*0.99++*	0.55+	1.00++	1.00++	0.43*	*1.00**	*1.00+*	*0.97++*	0.65++	1.00++	0.50*	0.43*
DP	***1.01+***	1.00*	***1.04****	0.33+	1.00++	0.20	0.25	1.00	1.00	*0.95++*	0.12*	1.00++	0.20	0.25	1.00	*1.00**	*0.90++*	0.47+	1.00++	0.20	0.25
FPGA	*1.00**	*0.99**	*0.79++*	0.61++	1.00++	0.50*	0.25	*1.00*	*1.00**	*0.81++*	0.90++	1.00++	1.00++	0.67+	*0.99+*	*0.96++*	*0.66++*	0.83++	1.00++	0.50*	0.43*
MP	*1.00+*	*1.00**	***1.06+***	0.61++	1.00++	0.50*	0.25	*1.00**	1.00	*0.98++*	0.46+	1.00++	0.20	0.25	*1.00+*	1.00	*0.95++*	0.63++	1.00++	0.20	0.11
CP	1.00	1.00	*0.97**	0.70++	1.00++	0.50*	0.43*	1.00	*1.00**	*0.97++*	0.73++	1.00++	1.00++	0.43*	*1.00*	*0.99**	*0.97++*	0.72++	1.00++	0.20	0.43*

Cohen's d: Negligible if $|d| \leq 0.147$, Small+ if $0.147 < |d| \leq 0.33$, Medium++ if $0.33 < |d| \leq 0.474$, **Large** if $|d| > 0.474$.
Wilcoxon Test: *Significant* if $p - value \leq 0.05$
Kendall's Tau (k): Weak* if $|\tau| \leq 0.3$, Moderate+ if $0.3 < |\tau| \leq 0.6$, Strong++ if $|\tau| > 0.6$.
Top-K (Tk: T1, T2, T3) Overlap: Negligible if $0.0 < Tk \leq 0.25$, Small* if $0.25 < Tk \leq 0.50$, Medium+ if $0.50 < Tk \leq 0.75$, Large++ if $0.75 < Tk \leq 1$.

Taken together, these findings show that the impact of MR deviations on peer review effort is context dependent. Deviations do not impact all MR types in the same way, and their impact also depends on how effort is measured. Their impact is limited and sometimes negative when effort is approximated through *time to complete peer review*, particularly for configuration MRs. In contrast, their impact becomes much more substantial when effort is measured through *amount*

Table 7.9 Performance and feature importance metrics for development MRs (time to complete peer review prediction)

Model	XGBoost							ExtraTree							Random Forest						
	Performance			Feature Importance				Performance			Feature Importance				Performance			Feature Importance			
	MSE	MAE	SA	K	T1	T3	T5	MSE	MAE	SA	K	T1	T3	T5	MSE	MAE	SA	K	T1	T3	T5
PF	*1.00**	*1.00**	0.99	0.47+	1.00++	0.50*	0.43*	1.00*	1.00	*0.99++*	0.66++	1.00++	1.00++	0.43*	1.00	*1.00**	*0.99++*	0.81++	1.00++	1.00++	0.67+
DP	***1.01****	1.00	*0.94++*	0.41+	1.00++	0.50*	0.43*	1.00	*1.00**	*0.99++*	0.76++	1.00++	0.50*	0.67+	1.00	*1.00**	*0.99++*	0.81++	1.00++	0.50*	0.67+
FPGA	*1.00+*	*1.00+*	**1.02**	0.55+	1.00++	0.20	0.25	*1.00+*	*1.00**	*0.93++*	0.42+	1.00++	0.20	0.25	*1.00++*	***1.01+***	*0.89++*	0.72++	1.00++	0.20	0.43*
MP	*1.00+*	*1.00+*	***1.03++***	0.69++	1.00++	0.20	0.25	*1.00+*	*1.00**	1.00	0.51+	1.00++	0.20	0.25	*1.00++*	*1.00**	*1.00**	0.76++	1.00++	0.20	0.43*
CP	*1.00*	*1.00**	***1.01***	0.67++	1.00++	0.50*	0.25	*1.00**	*1.00+*	*1.00+*	0.72++	1.00++	0.50*	0.43*	*1.00**	*1.00+*	*1.00++*	0.84++	1.00++	0.20	0.43*

Cohen's d: Negligible if $|d| \leq 0.147$, Small+ if $0.147 < |d| \leq 0.33$, Medium++ if $0.33 < |d| \leq 0.474$, **Large** if $|d| > 0.474$.
Wilcoxon Test: *Significant* if $p - value \leq 0.05$
Kendall's Tau (k): Weak* if $|\tau| \leq 0.3$, Moderate+ if $0.3 < |\tau| \leq 0.6$, Strong++ if $|\tau| > 0.6$.
Top-K (Tk: T1, T2, T3) Overlap: Negligible if $0.0 < Tk \leq 0.25$, Small* if $0.25 < Tk \leq 0.50$, Medium+ if $0.50 < Tk \leq 0.75$, Large++ if $0.75 < Tk \leq 1$.

of changes required to complete peer review, where removing deviations frequently improves predictive performance and alters model interpretation for both configuration and development MRs. This result reinforces the broader argument of this RQ: deviation-aware analysis should not only be performed globally, but should also account for MR type, since both the prevalence and the analytical consequences of deviations vary across artifact contexts.

Table 7.10 Performance and feature importance metrics for configuration MRs (amount of changes required to complete peer review prediction)

Model	XGBoost							ExtraTree							Random Forest						
	Performance			Feature Importance				Performance			Feature Importance				Performance			Feature Importance			
	MSE	MAE	SA	K	T1	T3	T5	MSE	MAE	SA	K	T1	T3	T5	MSE	MAE	SA	K	T1	T3	T5
PF	**1.11**	***1.36+***	***1.56++***	0.56+	0	0.50*	0.67+	1.00	1.00*	*0.99++*	0.55+	1.00++	1.00++	0.43*	**1.12**	***1.38++***	*0.46++*	0.69++	0	0.20	0.43*
DP	***4.25++***	***2.69++***	***2.04++***	0.54+	0	0.20	0.43*	1.00	1.00	*0.95++*	0.12*	1.00++	0.20	0.25	***4.44++***	***2.93++***	*0.50++*	0.79++	0	0.50*	0.67+
FPGA	**1.18**	**1.09**	**1.03**	0.75++	1.00++	0.20	0.43*	*1.00*	*1.00**	*0.81++*	0.90++	1.00++	1.00++	0.67+	0.90	**1.06**	*0.93**	0.79++	1.00++	0.20	0.25
MP	***2.83++***	***2.09++***	***1.46++***	0.62++	0	0.50*	0.43*	*1.00**	1.00	*0.98++*	0.46+	1.00++	0.20	0.25	***2.88++***	***2.14++***	*0.44++*	0.81++	0	0.50*	0.43*
CP	***1.59+***	***1.44++***	***1.39++***	0.65++	0	0.50*	0.43*	1.00	*1.00**	*0.97++*	0.73++	1.00++	1.00++	0.43*	***1.54+***	***1.51++***	*0.61++*	0.88++	0	0.50*	0.67+

Cohen's d: Negligible if $|d| \leq 0.147$, Small+ if $0.147 < |d| \leq 0.33$, Medium++ if $0.33 < |d| \leq 0.474$, **Large** if $|d| > 0.474$.
Wilcoxon Test: *Significant* if $p - value \leq 0.05$
Kendall's Tau (k): Weak* if $|\tau| \leq 0.3$, Moderate+ if $0.3 < |\tau| \leq 0.6$, Strong++ if $|\tau| > 0.6$.
Top-K (Tk: T1, T2, T3) Overlap: Negligible if $0.0 < Tk \leq 0.25$, Small* if $0.25 < Tk \leq 0.50$, Medium+ if $0.50 < Tk \leq 0.75$, Large++ if $0.75 < Tk \leq 1$.

7.4 Discussion

Generalizability

(a) Industry: To validate our findings and ensure the robustness of our approach, we solicited data from TELUS, which allows us to extend our analysis beyond our initial dataset. Following

Table 7.11 Performance and feature importance metrics for development MRs (amount of changes required to complete peer review prediction)

Model	XGBoost							ExtraTree							Random Forest						
	Performance			Feature Importance				Performance			Feature Importance				Performance			Feature Importance			
	MSE	MAE	SA	K	T1	T3	T5	MSE	MAE	SA	K	T1	T3	T5	MSE	MAE	SA	K	T1	T3	T5
PF	***1.90+***	***1.45++***	***1.24++***	0.59+	0	0.50*	0.25	1.00*	1.00	*0.99++*	0.66++	1.00++	1.00++	0.43*	***1.83+***	***1.55++***	*0.70++*	0.85++	0	0.50*	0.67+
DP	***1.78++***	***1.33++***	***1.52++***	0.56+	0	0.20	0.43*	1.00	*1.00**	*0.99++*	0.76++	1.00++	0.50*	0.67+	***1.97++***	***1.54++***	*0.71++*	0.78++	0	0.20	0.25
FPGA	*0.30**	*0.80**	**1.02**	0.67++	0	0.50*	0.67+	*1.00+*	*1.00**	*0.93++*	0.42+	1.00++	0.20	0.25	*0.44**	0.87	*0.63+*	0.82++	0	0.50*	0.67+
MP	***2.11++***	***1.66++***	0.99	0.54+	0	0.50*	0.25	*1.00+*	*1.00**	1.00	0.51+	1.00++	0.20	0.25	***2.16++***	***1.80++***	*0.61++*	0.83++	0	0.50*	0.67+
CP	***1.81++***	***1.56++***	0.99	0.57+	0	0.50*	0.67+	*1.00**	*1.00+*	*1.00+*	0.72++	1.00++	0.50*	0.43*	***2.03++***	***1.76++***	*0.63++*	0.84++	0	0.20	0.67+

Cohen's d: Negligible if $|d| \leq 0.147$, Small+ if $0.147 < |d| \leq 0.33$, Medium++ if $0.33 < |d| \leq 0.474$, **Large** if $|d| > 0.474$.
Wilcoxon Test: *Significant* if $p - value \leq 0.05$
Kendall's Tau (k): Weak* if $|\tau| \leq 0.3$, Moderate+ if $0.3 < |\tau| \leq 0.6$, Strong++ if $|\tau| > 0.6$.
Top-K (Tk: T1, T2, T3) Overlap: Negligible if $0.0 < Tk \leq 0.25$, Small* if $0.25 < Tk \leq 0.50$, Medium+ if $0.50 < Tk \leq 0.75$, Large++ if $0.75 < Tk \leq 1$.

the same process of MR annotation, we collaborate with their industry experts to identify deviations in one of their most critical projects, which contains 9k MRs.

Our analysis revealed that the same deviations occurred in 17.88% of cases. As shown in Figure 7.6, we observe a similar pattern to our previous findings, where Library Updates (LU) accounted for 39.1% of the deviations, followed by Build or Configuration Adjustments (BOCA) at 39.1%. This high proportion can be attributed to the nature of the company's development process. The industry expert played a crucial role in distinguishing between normal cases and those that did not require review during dataset annotation. These findings emphasize the importance for practitioners to analyze similar patterns in their own datasets, as they may influence review analysis and related decisions.

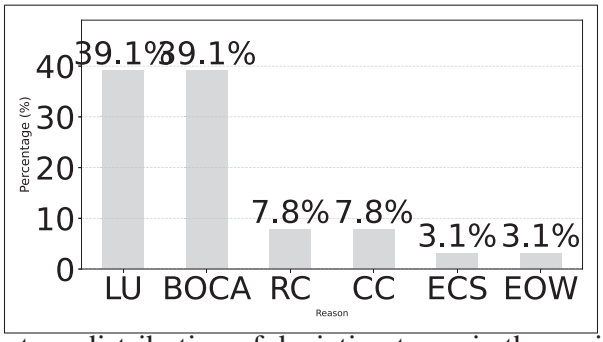


Figure 7.6 Percentage distribution of deviation types in the project from the other industrial partner *Note: BOCA = Build or Configuration Adjustments, EOW = Experimental or Work in Progress, CC = Code Cleaning, LU = Library Update, RC = Revert Commits, HC = Huge Changes, ECS = Empty Change Set.*

(b) Open Source: Analyzing deviations in open-source projects is challenging due to the lack of direct industrial partners for validation. In industry, deviations can be confirmed through collaboration with practitioners who provide context on project workflows and MR usage. In contrast, open-source projects follow a decentralized model with diverse contributors, making validation dependent on metadata, commit messages, and review patterns. Without direct access to maintainers, distinguishing true deviations from standard variations becomes more difficult.

Despite these challenges, applying our methodology to an open-source project is valuable, as it allows us to assess the generalizability of deviation detection beyond controlled industrial environments. By replicating our approach, we analyze the Inkscape project, which includes 6,989 MRs, making it a suitable case study due to its active community-driven development and diverse MR usage. As shown in Figure 7.7, our analysis shows that deviations account for 10.69% of cases, though not all deviation types are equally prominent. Notably, experimental or WIP MRs represent the largest share (43.2%) [2] [3], followed by empty change set (29.7%) [4] and code cleaning (24.3%)[5]. Revert commits, on the other hand, are relatively rare, comprising only 2.7% of deviation cases [6]. Interestingly, an additional deviation type emerges in this context: **Documentation Updates**, which include MRs related to documentation modifications and language translations where we do not observe any review activity[7] [8]. With this new deviation, MR deviations account for 16% of total deviations. ***This finding highlights the importance of investigating deviation patterns within different contexts, encouraging practitioners to examine deviations specific to their own projects.*** Notably, The overall percentage of deviations in Inkscape is lower than in the industrial dataset, which we attribute to differences in workflow structures, review policies, and project governance. In open-source projects, we hypothesize that review deviations are naturally less frequent due to their more flexible and asynchronous review

[2] https://gitlab.com/inkscape/inkscape/-/merge_requests/2467

[3] https://gitlab.com/inkscape/inkscape/-/merge_requests/747

[4] https://gitlab.com/inkscape/inkscape/-/merge_requests/457

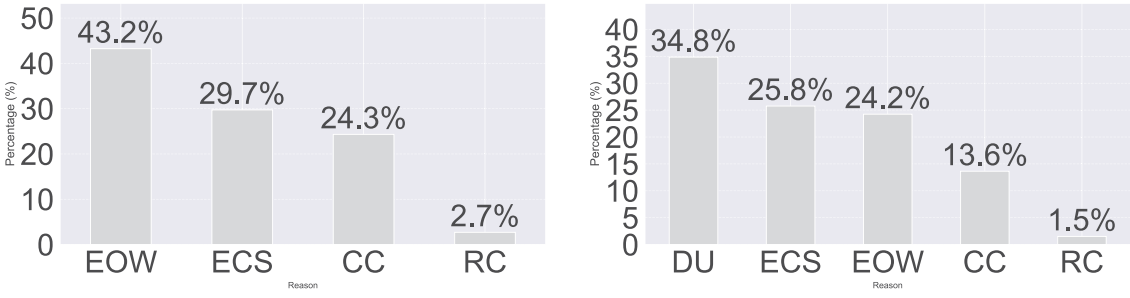
[5] https://gitlab.com/inkscape/inkscape/-/merge_requests/3475

[6] https://gitlab.com/inkscape/inkscape/-/merge_requests/3342

[7] https://gitlab.com/inkscape/inkscape/-/merge_requests/2295

[8] https://gitlab.com/inkscape/inkscape/-/merge_requests/33264

processes, whereas in industrial settings, structured workflows and predefined roles contribute to a higher incidence of deviations.



a) Distribution of MR deviations before adding the new category

b) Distribution of MR deviations after adding the new category

Figure 7.7 Percentage distribution of deviation types for Inkscape

Note: BOCA = Build or Configuration Adjustments, EOW = Experimental or Work in Progress, CC = Code Cleaning, LU = Library Update, RC = Revert Commits, HC = Huge Changes, ECS = Empty Change Set, DU = Documentation Updates.

Explaining MR deviations: The MR deviations identified in RQ1 do not conform to the standard peer review process, with each type of deviation exhibiting distinct characteristics and behaviors. These deviations account for up to 37.02% of the total MRs, potentially distorting key insights into review dynamics. Understanding these deviations individually provides critical supplementary information that would otherwise remain hidden. Ignoring these MRs would exclude a substantial portion of project activity, which is why we conducted additional analyses to investigate their interpretation compared to standard MRs.

From a statistical perspective, an analysis of time to complete peer review did not reveal a significant difference between standard and deviation MRs. However, ***our machine learning experiments with 100 bootstraps uncover notable distinctions in feature importance***. Specifically, when comparing models trained on deviation MRs versus regular MRs, XGBoost (the best-performing model in RQ3, with results consistent across other algorithms) exhibits distinct behaviors. The Kendall's tau correlation between models trained on standard and deviation MRs ranges from 0.26 to 0.64, indicating varying degrees of divergence. Additionally, while Top-1 feature rankings remain stable across all projects, Top-3 feature overlap is 0.5 in

four projects and 0.2 in one project. Finally, Top-5 feature overlap is 0.25 in three projects and 0.42 in two projects, highlighting the unique patterns embedded in MR deviations compared to regular MRs. The observed differences suggest that treating deviation MRs separately in predictive models could enhance peer review analysis by introducing an additional analytical dimension, enabling deeper insights from machine learning models.

Model Validation: To assess the robustness and generalizability of our predictive models, we conducted cross-validation across different projects, training on one project (e.g., CP) and validating on another (e.g., DP). This approach evaluates the model's ability to adapt to unseen data and ensures that it is not overfitting to a specific project's characteristics. Our results indicate accuracy ranging from 0.54 to 0.72, highlighting the model's adaptability while also revealing variability in performance across different contexts.

Discussion Summary

This study shows that MR deviations are not merely isolated anomalies in the review process, but recurring forms of process variation that affect how MR data should be interpreted. The central implication is that an MR dataset should not be assumed to represent a homogeneous collection of standard review instances. In industrial DevOps settings, the MR mechanism can serve multiple purposes: formal peer review, dependency maintenance, configuration adjustment, code cleanup, rollback, experimentation, or operational synchronization. These purposes leave similar traces in the same repository infrastructure, but they do not necessarily correspond to the same review intent or effort demand. Treating all of them as equivalent review units can therefore lead to misleading conclusions about review effort, reviewer behavior, and the factors that drive review outcomes.

MR Deviations as Workflow Signals Rather Than Statistical Noise

A key lesson from our results is that deviations should not be reduced to the notion of statistical outliers. Traditional preprocessing strategies often remove observations that are numerically

extreme, for example unusually long reviews, unusually large changes, or unusually inactive MRs. However, our findings suggest that this is not sufficient. Some deviations may look numerically ordinary while still reflecting a non-standard review workflow. Conversely, some regular MRs may appear large, slow, or inactive while still corresponding to legitimate peer-reviewed work.

This distinction matters because deviations are not only data quality problems; they are also workflow signals. They reveal how teams actually use the MR mechanism beyond its formal purpose. For example, library updates and build or configuration adjustments suggest that the MR mechanism is also used as a controlled integration gate for routine operational changes. Experimental or work-in-progress MRs show that developers sometimes use MRs as coordination or staging artifacts before a change is ready for full review. Code cleaning and revert-related MRs show that some MRs are created to maintain or restore the codebase rather than to introduce new functionality. These cases are meaningful for process understanding, but they should not always be analyzed together with regular review-intensive MRs.

Therefore, deviation-aware analysis should not be understood as a simple filtering step. Instead, it should be viewed as a way to recover the process intent behind MR traces. Depending on the research or managerial objective, deviations may need to be excluded, analyzed separately, or explicitly modeled. If the goal is to study standard peer review effort, excluding deviations can reduce bias. If the goal is to understand how teams use the MR mechanism in practice, deviations are themselves important process phenomena. This distinction helps avoid a simplistic interpretation where deviations are always treated as noise.

Why Deviations Affect Effort Modeling

The impact of deviations on predictive models can be explained by the fact that effort-related metrics have different meanings across standard and non-standard MRs. In regular MRs, completion time, post-submission changes, reviewer participation, and discussion activity are more likely to reflect the actual cost of evaluating and improving a submitted change. In

deviations, the same metrics may instead reflect procedural waiting time, automation behavior, synchronization constraints, or low-priority maintenance activity.

This difference changes both model performance and model interpretation. When deviations are mixed with regular MRs, the model is forced to learn from cases generated by different underlying processes. As a result, the relationship between predictors and review effort becomes less stable. A feature that appears important in the full dataset may be important because it captures deviation-related behavior rather than standard review effort. After deviations are removed, the model may rely on a different set of features, indicating that the original interpretation was partly driven by non-standard cases.

This is especially important for empirical software engineering studies that use feature importance to explain review behavior. In such studies, the objective is often not only to predict an outcome, but also to understand what drives that outcome. Our results show that even when predictive performance changes only moderately, feature importance rankings can shift substantially. This means that deviations may have a stronger effect on explanation than on prediction. For researchers, this is an important warning: a model can appear reasonably accurate while still producing an interpretation that is sensitive to the composition of the dataset.

The Multi-Dimensional Nature of Review Effort

Another important implication is that the effect of deviations is not limited to temporal effort. peer review effort is often approximated through completion time because time is easy to extract and has practical relevance. However, completion time mixes several phenomena: active review, waiting, prioritization, scheduling, and coordination delays. The amount of changes required to complete an MR captures a different dimension of effort, namely how much the submitted change evolves after it enters the review process.

By showing that deviations affect both completion time and the amount of changes required to complete an MR, our study indicates that their impact is multi-dimensional. Deviations do not only distort how long reviews appear to take; they also affect how much modification

effort appears to be associated with MR completion. This reinforces the need to analyze review effort through complementary measures. A deviation may have a short completion time but no meaningful review activity, or it may remain open for a long time because it is low priority rather than difficult to review. Similarly, a regular MR and a deviation may have similar change sizes, while reflecting different review purposes.

This result supports a broader view of review effort analysis. Instead of relying on a single effort proxy, researchers should consider whether each metric reflects the same process construct across all MRs. Completion time, amount of changes, discussion activity, reviewer participation, and approval behavior may each be affected differently by deviations. A deviation-aware approach can therefore improve construct validity by ensuring that effort measures are interpreted in relation to the type of MR workflow they represent.

MR Type Changes the Meaning of Deviations

The MR-type-aware analysis further shows that deviations are not uniformly distributed across artifact types. This matters because artifact type changes the interpretation of both the deviation and the effort measure. A configuration MR that updates deployment scripts, dependency versions, or CI/CD settings may formally appear as a configuration-related review instance, but its actual review intent may be procedural or operational. Similarly, a development MR may correspond to implementation work, but it may also represent cleanup, rollback, or experimentation.

This distinction is important for studies comparing configuration, development, and documentation MRs. Without deviation awareness, one may incorrectly attribute differences in review effort to artifact type, when part of the difference is actually due to the prevalence of specific deviation categories. For example, if configuration MRs contain many library updates or build adjustments, they may appear to require less review effort not only because configuration changes are reviewed differently, but also because many of these MRs correspond to lightweight procedural cases.

Conversely, development MRs may show more heterogeneous effort patterns because they include a wider variety of deviation types.

The broader implication is that artifact type and deviation status should be considered jointly. MR type explains what kind of artifact is being modified, whereas deviation status explains whether the MR follows the expected review workflow. Both dimensions are necessary to interpret review effort correctly. An MR-type-aware but deviation-agnostic analysis may still mix regular and non-standard workflows. A deviation-aware but type-agnostic analysis may miss the fact that deviations have different forms and consequences depending on the artifact being reviewed.

From Deviation Removal to Deviation-Aware Analytics

Although our empirical analysis compares models trained with and without deviations, the practical objective is not to recommend the systematic removal of all deviations from MR datasets. The appropriate treatment depends on the purpose of the analysis. For studies of standard review effort, removing or separately modeling deviations can improve analytical validity. For studies of process governance, deviations may be valuable because they reveal how teams use the MR mechanism for maintenance, coordination, and operational control. For industrial dashboards, deviations may need to be flagged rather than removed, so that teams can interpret metrics according to MR intent.

This suggests a shift from generic MR analytics to deviation-aware MR analytics. In such an approach, each MR would be analyzed not only through its size, duration, and review activity, but also through its likely process role. This could support more reliable dashboards, more precise effort estimation, and better reviewer allocation. For example, a routine dependency update and a complex feature implementation should not necessarily be treated as equivalent units when computing average review effort or identifying bottlenecks. Similarly, a long-open experimental MR should not be interpreted in the same way as a long-open feature MR waiting for reviewer feedback.

At the same time, deviation-aware analytics should be used carefully. The identification of a deviation does not imply that review should be skipped automatically. Some procedural or maintenance-oriented MRs may still carry operational risk, especially in configuration-heavy DevOps environments. The practical value of deviation detection is therefore not to bypass review, but to support more informed review triage. Teams can use deviation status as one signal among others to decide whether an MR requires standard review, lightweight review, automated validation, or additional scrutiny.

7.5 Implications

7.5.1 Implications for our industrial partners

For our industrial partners, these findings suggest that not all MRs should be treated as equally meaningful review units. In industrial settings, a portion of MR traffic may correspond to procedural, maintenance-oriented, or exploratory activities that formally use the MR mechanism without requiring the same level of reviewer attention as regular code changes. Identifying such cases can help teams allocate review effort more effectively and reduce unnecessary review overhead.

The results also indicate that deviation-aware analysis can improve the reliability of internal dashboards, monitoring systems, and predictive tools. When deviations remain mixed with regular MRs, indicators such as time to complete peer review, estimated effort, or feature importance may reflect a mixture of fundamentally different workflows. This can lead teams to optimize the wrong aspects of the process or misinterpret the main drivers of review effort. Integrating automated deviation detection into analytics pipelines could therefore support more reliable monitoring, more informed reviewer assignment, and more accurate process assessment.

Finally, our findings highlight the importance of considering MR type when interpreting analytics. This is especially relevant in DevOps-intensive environments where configuration-related changes are frequent and operationally central. Although such MRs may appear to

deserve systematic review attention, many of them may correspond to lightweight procedural updates. Distinguishing these cases from truly review-intensive changes can help teams better align review practices with actual effort and risk.

7.5.2 Implications for researchers

For researchers, the primary implication of this chapter is that MR-based empirical studies should treat deviation detection as a data-preparation step rather than as an optional refinement. Because deviations account for up to 37% of MRs in our industrial data and follow a workflow that differs fundamentally from the standard review–revise–approve–merge process, their inclusion in empirical analyses silently mixes heterogeneous behaviors under a single unit of analysis. The observed shifts in feature-importance rankings when deviations are removed indicate that models trained on unfiltered MR data may recover partially spurious drivers of peer review effort. Future empirical studies that aim to characterize peer review effort, identify review bottlenecks, or build predictive models of review outcomes should therefore explicitly report whether deviations were filtered, describe the criteria used for filtering, and, where possible, report sensitivity analyses comparing results with and without deviations. This expectation applies symmetrically to time-based and change-based effort measures.

The chapter also provides reusable methodological artifacts for the research community. The seven-category deviation taxonomy offers a shared vocabulary for describing non-standard MRs and can be extended or adapted to new contexts, while the few-shot detection method, which reaches up to 91% accuracy, demonstrates that deviation annotation can be scaled to large MR datasets without requiring exhaustive manual labeling. Both artifacts can support replication and cross-study comparability in empirical peer review research. Finally, the uneven distribution of deviations across configuration, development, and documentation MRs reinforces the point made in prior chapters that peer review should be analyzed through stratified rather than aggregate designs: deviations and MR types interact, and ignoring this interaction risks producing findings that do not hold uniformly across artifact categories. Researchers are therefore encouraged to

combine deviation-aware filtering with MR-type stratification when studying review effort, and to publish the filtering decisions alongside their replication packages.

7.6 Threats to Validity

This study focuses on the MR mechanism as implemented in GitLab. Although other review mechanisms, such as Pull Requests in GitHub, rely on similar principles of discussion, feedback, and approval, differences in workflow and tooling may affect the generalizability of our findings across platforms.

A second threat concerns the specificity of the analyzed projects. The prevalence and nature of deviations may vary depending on development practices, team structures, and domain-specific workflows. To mitigate this threat, the projects were selected in collaboration with industry experts and span multiple teams and types of work. Nevertheless, further replication in other industrial and open-source settings is needed to assess the generalizability of both the deviation taxonomy and the observed analytical effects.

A third threat lies in the technologies used for data processing and model training. We mitigated this risk by relying on widely adopted and well-maintained tools such as `scikit-learn`, which supports reproducibility, transparency, and robustness.

A fourth threat relates to the manual annotation process used to identify deviations. This process may introduce subjectivity. To reduce this risk, we involved domain experts in the annotation of industrial MRs and established annotation guidelines to improve consistency.

7.7 Conclusion

This study examined MRs that do not follow the expected review workflow, which we refer to as deviations. Through an industrial investigation, we showed that such cases are not marginal exceptions, but recurrent occurrences that account for up to 37.02% of MRs. Their prevalence alone indicates that treating all MRs as equivalent review instances can distort empirical analysis.

To address this issue, we first defined and categorized deviations, then proposed an AI-based detection approach using few-shot learning, achieving up to 91% accuracy while substantially reducing the manual effort required to identify them.

We further showed that deviations have a measurable impact on code review effort analysis. When effort is approximated through time to complete peer review, excluding deviations often improves or preserves model performance while substantially altering model interpretation. Extending the analysis to the amount of changes required to complete peer review confirmed that this effect is not limited to temporal measures, but also applies to a second, change-based effort dimension. Together, these findings show that deviations affect both how effort is predicted and how it is explained.

The study also showed that deviations are distributed differently across MR types. Configuration MRs are mainly associated with procedural deviations such as library updates and build or configuration adjustments, whereas development MRs exhibit a more heterogeneous profile, including code cleaning, experimental work, and revert-related cases. In addition, the analytical effect of excluding deviations differs across configuration and development MRs, especially when effort is measured through the amount of changes required to complete peer review. This indicates that deviation-aware analysis should account not only for whether an MR is standard or non-standard, but also for the type of artifact under review.

Overall, our results demonstrate that deviations constitute a meaningful source of bias in code review analytics. Ignoring them can lead to misleading interpretations of review effort and unreliable conclusions for both researchers and practitioners. By identifying deviations, automating their detection, and examining their impact across effort dimensions and MR types, this work contributes to more reliable and context-aware MR analytics. In practice, integrating deviation-aware analysis into review pipelines can help teams better prioritize reviewer attention and avoid overestimating the importance of lightweight procedural changes.

Acknowledgement: We extend our sincere gratitude to **Seyedbehnam Mashari** for their invaluable contribution and collaboration. Their expertise in MR analysis greatly enhanced the

reliability of our annotation phase. We also acknowledge **Bassem Guendy** for their continuous support and guidance throughout this study. Their expertise was instrumental in providing critical insights in our work, facilitating data access, and assisting in the validation of our results, ensuring the robustness of our analysis.

CHAPTER 8

DO SIZE ESTIMATES PREDICT DELIVERY? AN EMPIRICAL ANALYSIS OF TASK ESTIMATION IN INDUSTRIAL DEVOPS

Samah Kansab[1] , Damien Desvent[1] , Francis Bordeleau[1] , Ali Tizghadam[2]

[1] Department of Software Engineering and IT, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[2] TELUS, 25 York St, Toronto, Ontario, Canada M5J 2V5

Paper Accepted in the 42nd IEEE International Conference on Software Maintenance and Evolution on June 2026

Preamble

This chapter connects peer review effort to upstream planning conditions by studying size-based task estimation in an industrial DevOps environment at TELUS. We analyze whether estimated task sizes remain meaningful during execution, how waiting states and multitasking relate to task outcomes, how often underestimation contributes to failure, and whether task failure can be anticipated at creation time. The results show that failure rates remain relatively stable across sizes and over time, while multitasking has no practically meaningful relationship with failure. Waiting exposure is associated with higher failure rates, but waiting duration itself is less informative. Underestimation explains a meaningful share of unsuccessful outcomes, indicating that part of the process burden later observed downstream may originate from upstream planning inaccuracies. A bridge analysis further shows that underestimated failed tasks spend a disproportionate share of their lead time in review, suggesting that unresolved complexity may be transferred into the peer review stage.

This chapter is adapted from a paper submitted to ICSME 2026. The baseline of this work was developed in the master's thesis of Damien Desvent, who carried out the initial Kanban analysis. The thesis version revises the introduction, conclusion, and overall framing to align the chapter with the objectives of this thesis. It also extends the discussion by connecting the upstream-planning findings to peer review effort and to the broader process-oriented argument developed across the thesis.

8.1 Introduction

The previous chapters examined peer review effort from within the MR process itself, focusing on how review activity can be observed, how effort can be measured, how it varies across types of changes, and how non-standard workflows can affect its interpretation. Together, these analyses show that peer review effort is not a single or uniform phenomenon, but a process-dependent construct shaped by the characteristics and context of the reviewed change.

However, peer review effort does not begin only when an MR is created. The work required during review may already be shaped by upstream conditions, including how tasks are estimated, assigned, interrupted, and executed before they reach the review stage. A poorly estimated or unstable task may result in an MR that is less mature at submission time, requires more clarification, or generates more corrective work during review. Therefore, optimizing peer review effort requires looking not only at the review process itself, but also at the planning and execution conditions that may generate avoidable downstream burden.

Effort estimation plays a central role in software project planning, task allocation, and delivery monitoring, as it enables teams to anticipate the work required to complete tasks, assign responsibilities, and adjust priorities throughout development (Mahmood, Kama, Azmi, Khan & Ali, 2022; Pasuksmit, Thongtanunam & Karunasekera, 2024). When estimation and planning are inadequate, projects are more likely to experience delays or failure (Serrador & Pinto, 2015), a challenge that has intensified as software systems and delivery pipelines have grown in scale and complexity (**?**). In DevOps environments, where work is delivered continuously through platforms such as GitLab or GitHub, estimation supports rapid coordination and scheduling decisions rather than fixed upfront planning. For these reasons, estimation is a foundational practice for effective software delivery (Robles, Lara, Salgado & Hidalgo-Reyes, 2023).

Despite its importance, estimation remains difficult across Agile and flow-based planning approaches, including Scrum, Kanban, and related iterative methods (Ruk *et al.*, 2025b). Existing research has mainly examined estimation techniques such as story points, T-shirt sizing,

relative calibration, and model-based effort prediction (Mallidi & Sharma, 2021a; Sahni; Poženel *et al.*, 2023; Catak *et al.*, 2024; William *et al.*, 2021a; Thukkaram, 2025). However, less is known about how size-based estimates behave once tasks enter execution workflows in industrial DevOps settings. In particular, limited evidence exists on how estimates relate to observed lead time, execution interruptions, waiting states, developer workload, and task failure during day-to-day development (Weigelt; Alturki & E-amin, 2025).

This gap matters for the present thesis because upstream estimation conditions may influence downstream peer review effort. Inaccurate estimates can create unrealistic delivery expectations, suboptimal task allocation, and unstable execution conditions. These conditions may affect the maturity of the resulting change before review, increasing the need for clarification, revision, or coordination once the work is submitted as an MR. In this sense, part of peer review effort may be generated before the peer review stage begins, through upstream planning and execution dynamics.

This chapter therefore extends the thesis from the analysis of peer review effort inside the MR process to the upstream conditions that may shape it. We study this relationship in collaboration with TELUS, a major Canadian telecommunications company operating large-scale Kanban-based DevOps workflows on GitLab. In this environment, tasks are estimated using T-shirt sizes and progress through multiple workflow states without fixed iteration boundaries. This continuous-flow setting makes it possible to observe how estimates interact with execution dynamics over time, rather than treating estimation as a planning-only activity.

We analyze 13,419 tasks from 187 GitLab projects and reconstruct fine-grained task histories using GitLab APIs. From these data, we derive task lead times and define two complementary size-specific reference indicators: a median-based reference lead time (RLT^{MED}), which captures typical delivery behavior, and a percentile-based reference lead time (RLT^{SLE}), aligned with the Kanban Service Level Expectation concept. Based on these references, we define task-level failure indicators that identify tasks whose execution exceeds size-specific expectations. We further define an underestimation indicator that captures tasks that would have succeeded if

estimated one size higher, and we examine how estimation outcomes relate to waiting states, workload distribution, and size-weighted multitasking. Finally, we assess whether task failure can be predicted at creation time using supervised classifiers and locally deployed small language models.

Across this analysis, we find that task failure rates remain relatively stable across sizes and over time, with no clear long-term improvement or degradation trend. Waiting exposure is strongly associated with failure: tasks that enter a waiting state fail 1.5 to 2 times more often than tasks overall, although waiting duration itself does not significantly distinguish failed from successful tasks. Underestimation explains a meaningful but partial share of failures, ranging from 17.9% to 24.6% depending on the reference indicator, with stronger effects concentrated at middle and upper size transitions. Size-weighted multitasking shows no practically meaningful relationship with task failure, suggesting that parallel work in the TELUS setting reflects workload organization rather than a direct cause of unsuccessful delivery. Creation-time prediction is feasible but weak, with the strongest configurations reaching an AUC of 0.57, confirming that execution-time signals carry substantially more predictive power than creation-time features alone.

The main contributions of this chapter are:

- an execution-centric empirical analysis of size-based estimation in a Kanban-based industrial DevOps environment, focusing on estimation behavior during task execution rather than planning alone;
- a unified task-level framework that defines estimation failure using two complementary size-specific reference indicators, quantifies underestimation through minimal one-size corrections, and relates estimation outcomes to workflow dynamics such as waiting states and developer workload;
- an evaluation of early-warning prediction of task failure at creation time using supervised classifiers and locally deployed small language models, providing a benchmark for on-premises deployment in industrial settings;

- empirical evidence showing that upstream planning and execution conditions can help explain where downstream process burden may originate, supporting the thesis argument that peer review effort should be analyzed within the broader DevOps workflow.

8.2 Study Design

This study analyzes effort estimation in TELUS GitLab repositories answering 5 RQs, through four steps: data collection, metric construction, reference lead time definition, and statistical testing.

8.2.1 Research Questions

The chapter is guided by the following five research questions:

RQ1 **How do task failure rates evolve across estimated sizes and over time?** This question examines whether size-based estimation provides stable delivery expectations during execution or whether failure patterns vary systematically across sizes and time.

RQ2 **How does time spent in waiting columns impact task outcomes?** This question studies whether workflow interruptions act as meaningful risk signals for delivery failure and whether waiting duration adds explanatory value beyond waiting exposure itself.

RQ3 **To what extent does underestimation contribute to task failure?** This question quantifies the role of estimation calibration in delivery failure by assessing whether failed tasks could have succeeded under a minimal upward size correction.

RQ4 **What is the impact of multitasking and workload distribution on task failure?** This question studies whether parallel work and workload intensity are associated with a higher probability of exceeding the expected delivery window.

RQ5 **To what extent can issue delivery failure be predicted at task creation?** This question evaluates early prediction using only point-in-time information available when the task

is created, and compares conventional machine-learning models with locally hosted LLM-based alternatives.

8.2.2 Data collection

The study is based on the issue and workflow-history dataset introduced in Section 3.5, which consists of issues, users, and full issue histories extracted from TELUS GitLab repositories. The reconstructed task timelines support the computation of lead time, workflow exposure, waiting duration, and workload-related measures used throughout the chapter.

8.2.3 Metrics collection

From the collected GitLab event streams, we derive analysis-ready metrics grouped by entity type (Table 8.1). The processing pipeline performs timestamp alignment, type normalization, missing-value handling, duplicate removal, assignment reconstruction, and automated-account identification. This produces consistent measurements of task execution, workflow transitions, estimation changes, and human involvement.

8.2.4 Estimation metrics

8.2.4.1 Effort (issue weight)

TELUS estimates effort using T-shirt sizes mapped to Fibonacci weights: $\{\text{XS}, \text{S}, \text{M}, \text{L}, \text{XL}, \text{XXL}\} \rightarrow \{1, 2, 3, 5, 8, 13\}$. In the remainder of the paper, *estimate*, *size*, and *category* refer to this mapped weight.

8.2.4.2 Lead time (LT)

For issue i, lead time is defined as:

$$\text{LT}_i = t_{\text{end}}(i) - t_{\text{start}}(i), \tag{8.1}$$

Table 8.1 Data entities and metrics derived from GitLab for our study

Entity	Analytical role	Attributes (definition)
Project Context	Defines the organizational and temporal scope of analysis	`id`— project identifier; `name`— project label; `name_with_namespace`— fully qualified project name; `namespace_id`— organizational unit; `created_at`— project start time; `last_activity_at`— most recent activity timestamp.
Task Descriptor	Captures static task properties used for estimation and grouping	`id`— task identifier; `project_id`— associated project; `title`— task summary; `description`— free-text content; `weight`— estimated task size; `state`— open or closed; `created_at`— task start time; `closed_at`— task completion time; `author_id`— task creator; `assignees`— current responsible developers; `cloned_from`— origin task identifier.
Actor Profile	Identifies human and automated contributors involved in task execution	`id`— actor identifier; `username`— account login; `name`— display name; `state`— account status; `is_bot`— automation indicator used to separate human and non-human activity.
Workflow Transition Event	Enables reconstruction of task movement across Kanban columns	`id`— event identifier; `user_id`— actor triggering the transition; `target_id`— task identifier; `created_at`— transition timestamp; `action`— label addition or removal; `label_id`, `label_name`— status labels encoding Kanban columns (e.g., `Status::*`).
Estimation Update Event	Tracks the evolution of task size estimates during execution	`id`— event identifier; `user_id`— actor modifying the estimate; `target_id`— task identifier; `created_at`— update timestamp; `weight`— new estimated size value.
Lifecycle State Event	Defines task activation and completion boundaries	`id`— event identifier; `user_id`— actor; `target_id`— task identifier; `target_type`— resource type; `created_at`— state-change timestamp; `state`— new lifecycle state (open or closed).
Assignment Change Event	Supports workload and multitasking measurement	`id`— event identifier; `target_id`— task identifier; `author_id`— actor; `assignee_id`— developer added or removed; `action`— assignment or unassignment; `created_at`— event timestamp.

where $t_{\text{start}}(i)$ is the first transition into `In Progress` and $t_{\text{end}}(i)$ is the first transition into `Done` or `Closed`.

8.2.4.3 Reference lead time (RLT) and outcome labeling

For each estimated size s, we define two complementary reference lead time indicators: an SLE-based threshold RLT_s^{SLE} and a median-based threshold RLT_s^{MED}.

SLE-based reference lead time (RLT_s^{SLE})

Following Kanban SLE practice (Vacanti & Coleman, 2020; Kan, 2022), we define:

$$\text{RLT}_s^{SLE} = Q_{0.85}\big(\{\text{LT}_i \mid \text{estimate}(i) = s\}\big), \tag{8.2}$$

Table 8.2 Per-size lead time distribution at TELUS. Values in hours. RLT^{MED} and RLT^{SLE} denote the median-based and SLE-based (85th percentile) reference lead times, respectively

Weight	T-shirt size	#Issues	$\mathrm{RLT}^{\mathrm{MED}}$	$\mathrm{RLT}^{\mathrm{SLE}}$	Mean	SD	Q1	Q3	Skew
0	NE	4,473	212.02	3,084.92	2,044.95	5,424.14	26.82	1,348.25	4.50
1	XS	4,311	195.30	837.65	584.98	1,465.05	65.64	482.80	7.56
2	S	798	307.94	1,020.88	655.36	1,185.66	97.52	673.99	4.61
3	M	2,422	341.28	1,320.13	829.16	1,560.93	189.00	816.04	6.00
5	L	982	526.78	1,893.77	1,182.57	2,021.81	310.24	1,181.66	4.73
8	XL	323	647.78	2,340.24	1,384.46	2,282.10	314.24	1,496.35	4.05
13	XXL	110	1,133.97	6,089.87	3,135.99	4,940.27	449.54	3,138.84	2.51

where $Q_{0.85}(\cdot)$ is the 85th percentile. This threshold captures a delivery-commitment view of expected lead time and is appropriate for our heavily right-skewed lead time distributions.

Under this definition, task i is labeled a ***success*** if $\mathrm{LT}_i \leq \mathrm{RLT}_s^{SLE}$ and a ***failure*** otherwise.

Median-based reference lead time (RLT_s^{MED})

As a complementary descriptive baseline, we define:

$$\mathrm{RLT}_s^{MED} = \mathrm{median}\big(\{\mathrm{LT}_i \mid \mathrm{estimate}(i) = s\}\big). \tag{8.3}$$

To reduce threshold noise, we apply a tolerance band $\tau = 0.40$:

$$\textbf{success} \text{ if } \mathrm{LT}_i < (1-\tau)\mathrm{RLT}_s^{MED}, \qquad \textbf{failure} \text{ otherwise} \tag{8.4}$$

while tasks within $[(1-\tau)\mathrm{RLT}_s^{MED},\ (1+\tau)\mathrm{RLT}_s^{MED}]$ are excluded from binary analyses (discretization noise (Rajbahadur *et al.*, 2019)).

Sensitivity to τ

We selected $\tau = 0.40$ because it lies near the knee of the failure-share curve between $\tau = 0.30$ and $\tau = 0.50$. Repeating the analyses with $\tau \in \{0.30, 0.50\}$ yields the same qualitative conclusions.

8.2.4.4 Reference lead time distribution

Table 8.2 reports both reference indicators together with the main distributional statistics. Of the 13,419 closed issues, 4,473 are non-estimated and excluded from size-dependent analyses, leaving 8,946 estimated issues. Smaller sizes dominate the dataset, especially XS. Both RLT^{MED} and RLT^{SLE} increase monotonically with size, consistent with the ordinal interpretation of T-shirt estimates. Lead times are strongly right-skewed across all sizes, which motivates the joint use of both indicators: RLT^{SLE} captures tail-based delivery commitment, whereas RLT^{MED} captures typical delivery and supports later adjustment analyses.

8.2.5 Statistical significance

We use non-parametric tests throughout because lead time, waiting percentage, and MV are strongly right-skewed. We apply the chi-square test of independence when both variables are categorical, the Mann–Whitney U test when comparing a continuous variable across failed and successful tasks, and Spearman rank correlation when testing monotonic trends between a continuous predictor and binary outcome. All tests use $\alpha = 0.05$.

8.3 Estimation Stability Across Time and Size

RQ1: How do task failure rates evolve over time and across estimated sizes

Approach

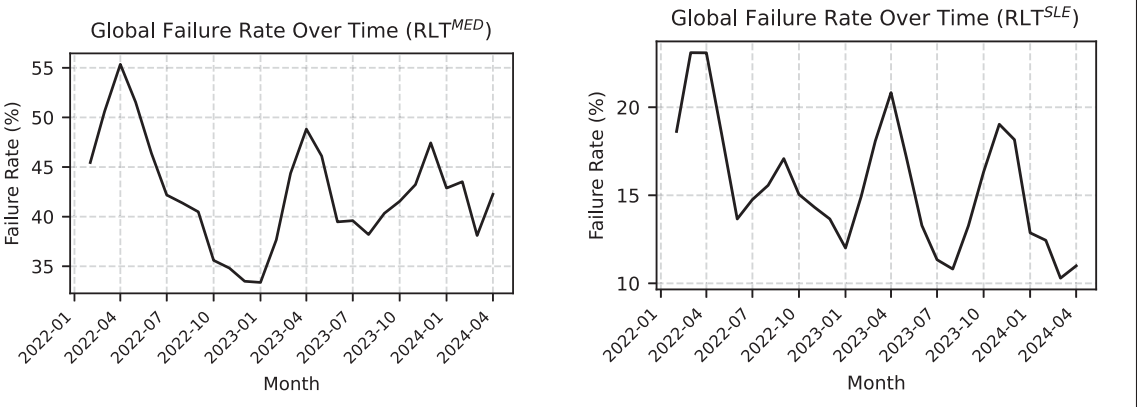
We define task failure by comparing each task's observed lead time with its size-specific reference lead time, as described in Section 8.2.

We first compute the failure rate for each estimated size to determine whether some size categories are more failure-prone than others. We then analyze monthly failure rates over the study period, both globally and by size category, to assess whether estimation performance exhibits long-term trends or only short-term fluctuations.

Results

Failure rates are remarkably uniform across estimated sizes. Under RLT^{SLE}, per-size failure rates are close to 15% by construction because the threshold is defined as the 85th percentile within each size category. Cross-size comparisons are therefore more informative under RLT^{MED}. As shown in Table 8.5, failure rates under RLT^{MED} range from 37.5% (S) to 43.0% (XS), a spread of only 5.5 percentage points. No size category stands out as substantially more failure-prone than the others, indicating that task size alone is not a determinant of estimation failure in this setting.

Failure rates remain stable over time under both indicators. Figure 8.1 shows the monthly global failure rate under both reference indicators. Under RLT^{MED}, the rate fluctuates between about 33% and 55%, whereas under RLT^{SLE} it varies between about 10% and 23%. Despite these different baseline levels, both curves follow the same overall pattern, with peaks and troughs occurring at the same periods. Neither indicator shows a sustained increase or decrease over time, suggesting that the observed variation reflects short-term contextual effects rather than progressive improvement or degradation in estimation performance.

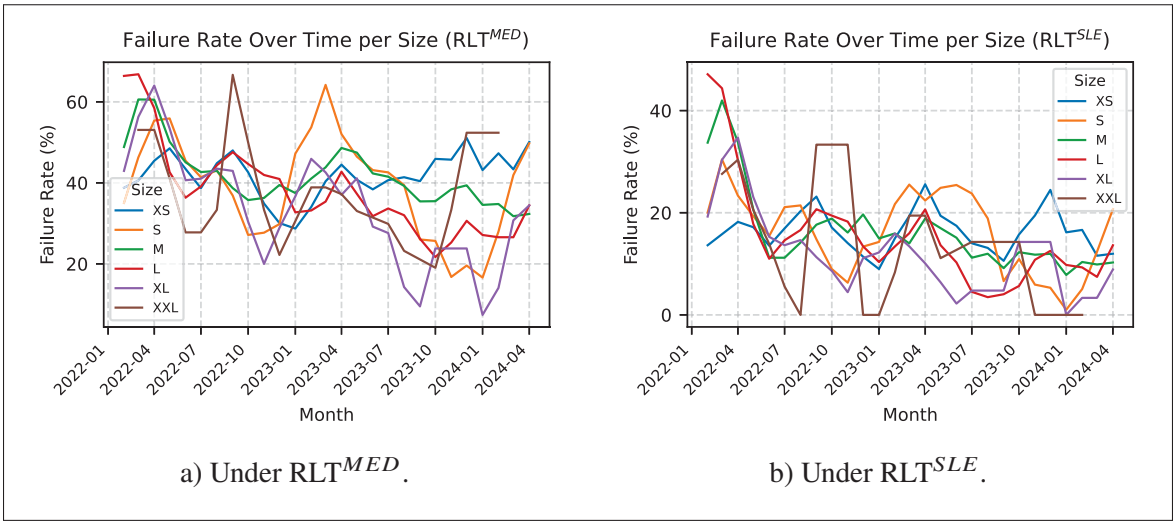


a) Under RLT^{MED}.

b) Under RLT^{SLE}.

Figure 8.1 Global failure rate over time (2022–2024) under both reference indicators

Temporal variation is comparable across size categories. Figure 8.2 shows that all size categories follow broadly similar temporal patterns under both RLT^{MED} and RLT^{SLE}. Short-lived increases and decreases appear at similar periods across sizes, and no category consistently performs better or worse than the others. The larger fluctuations observed for XXL under RLT^{SLE} are best explained by the small number of XXL tasks (110 in total) rather than by a substantive size-specific effect.



a) Under RLT^{MED}. b) Under RLT^{SLE}.

Figure 8.2 Failure rate over time per issue size (2022–2024) under both reference indicators

8.4 Waiting States and Task Outcomes

RQ2: How does time spent in waiting columns impact task outcomes

Approach

We study waiting from two complementary perspectives: *exposure* and *intensity*.

A task is considered exposed to waiting if it enters at least one of the following columns during its lifecycle: *Blocked*, *On Hold*, or *Waiting for External*. Using GitLab label-event histories, we determine whether each task ever entered one of these states.

To measure waiting intensity, we compute the total time spent in waiting columns and normalize it by task lead time. This yields the proportion of the task lifecycle spent waiting. We then compare failure rates between waiting and non-waiting tasks, and test whether failed waiting tasks spend a larger share of their lead time in waiting than successful waiting tasks.

Waiting Columns Definition

We consider a task to be in a *waiting state* whenever it is assigned one of the following Kanban labels:

- **Blocked**: work cannot proceed due to an unresolved issue;
- **On Hold**: work is intentionally paused;
- **Waiting for External**: progress depends on an external actor, team, or system.

These labels are not mutually exclusive over time. A task may enter more than one waiting state during its lifecycle.

Exposure to Waiting

For each closed issue i, we inspect its complete label-event history and define whether it was ever exposed to waiting. Let $\mathsf{Has}(i, k)$ be true if task i is assigned waiting label k at least once. We define:

$$\text{WaitExp}_i = \mathbf{1}[\exists\, k \in \mathcal{W} :\ \mathsf{Has}(i, k)]\,. \tag{8.5}$$

This indicator captures whether a task experienced at least one waiting interruption during execution.

Waiting Intensity

To quantify waiting intensity, we compute the total time spent in waiting columns:

$$\mathrm{TWC}_i = \sum_{k \in \{\texttt{Blocked}, \texttt{On Hold}, \texttt{Waiting for External}\}} \mathrm{dur}_i(k), \tag{8.6}$$

where $\mathrm{dur}_i(k)$ is the total duration during which waiting label k is active for task i.

We then normalize this value by lead time:

$$\mathrm{WaitingPct}_i = \frac{\mathrm{TWC}_i}{\mathrm{LT}_i}. \tag{8.7}$$

This ratio expresses how much of the task lifecycle is spent waiting.

Relating Waiting to Task Outcomes

We compare failure rates of waiting tasks with the overall failure rates reported in RQ1. We also compare WaitingPct between failed and successful tasks among those that experienced waiting. Failure is defined using the same size-specific reference thresholds introduced in Section 8.2.

Results

External-dependency waiting increases with task size. Table 8.3 shows that exposure to *Waiting for External* rises with size, reaching 10.6% for XL and 11.1% for XXL, compared with 4.9% for XS. Exposure to *Blocked* and *On Hold* is broadly comparable across sizes, with no clear monotonic trend. This pattern suggests that larger tasks are more likely to depend on cross-team coordination or external inputs, whereas internal interruptions affect tasks of all sizes similarly.

Waiting tasks fail substantially more often than tasks overall. Under both reference indicators, tasks that enter a waiting state have substantially higher failure rates than the overall rates reported

Table 8.3 Waiting-task statistics and failure rates among waiting tasks, by estimated size. Failure rates are computed on the subset of tasks that entered at least one waiting column, under both reference indicators (RLT^{MED} and RLT^{SLE})

Size	%_blocked	%_on_hold	%_waiting_ext	%_fail_MED	%_fail_SLE	mean_wait_pct	med_wait_pct	mean_wait_h	med_wait_h
XS	3.79	4.93	4.90	73.10	37.13	44.23	22.20	531.36	94.08
S	6.09	6.56	6.25	64.49	30.84	34.29	25.31	363.15	122.98
M	5.76	7.00	4.58	64.29	25.32	37.56	20.47	361.54	143.54
L	8.40	8.52	5.39	50.98	21.57	26.45	19.60	348.46	125.06
XL	6.04	9.06	10.57	58.49	30.19	34.79	27.47	1086.67	215.36
XXL	6.17	8.64	11.11	45.00	15.00	18.83	5.09	868.25	60.67

in RQ1. Under RLT^{MED}, failure among waiting tasks ranges from 45.0% to 73.1%, compared with a baseline of 37%–43%. Under RLT^{SLE}, it ranges from 15.0% to 37.1%, compared with a baseline of approximately 15%. Waiting tasks therefore fail about 1.5 to 2 times more often than tasks overall.

The effect of waiting is strongest for small and medium tasks. Although large tasks are more often exposed to waiting, the relative impact of waiting is stronger for smaller tasks. Under RLT^{MED}, the failure rate increases from about 44% to 73% for XS, from 37% to 64% for S, and from 41% to 64% for M. In contrast, waiting has a limited additional effect on XXL tasks, whose waiting-task failure rate remains close to the baseline. This suggests that smaller tasks have less tolerance for interruption, whereas larger tasks may absorb delays within a broader delivery window.

When waiting occurs, it occupies a substantial part of task execution. Table 8.3 also shows that waiting represents a meaningful share of lead time. For most sizes, the median waiting proportion ranges from about 19% to 27%. In absolute terms, XL and XXL tasks accumulate the highest waiting hours. However, XXL tasks have a much lower median waiting percentage (5.1%), consistent with the presence of administrative or quickly closed tasks in that category.

Waiting exposure predicts failure, but waiting duration does not. Table 8.4 confirms this distinction statistically. Chi-square tests reveal a significant association between waiting exposure and task failure under both indicators, both overall ($p < 0.001$) and for every size category except XXL, where the small sample ($n = 20$ waiting tasks) limits statistical power. In contrast, Mann–Whitney U tests show no significant difference in waiting percentage between

Table 8.4 Significance of the waiting–failure association. χ^2 tests compare failure rates of waiting vs. non-waiting tasks; Mann–Whitney U compares waiting percentage between failed and successful waiting tasks (p-values)

Test	Ind.	All	XS	S	M	L	XL	XXL	MW
χ^2	MED	<.001	<.001	<.001	<.001	<.001	.004	.560	.979
χ^2	SLE	<.001	<.001	<.001	<.001	.017	.001	–	.571

failed and successful tasks among tasks that already experienced waiting ($p = 0.979$ under RLT^{MED}, $p = 0.571$ under RLT^{SLE}). The predictive signal is therefore whether a task enters a waiting state, not how long it remains there.

Overall, RQ2 shows that waiting is not merely a descriptive workflow condition. Entering a waiting column is a strong signal that a task is at higher risk of exceeding its expected lead time, yet once a task waits, the duration of that waiting does not further distinguish failed from successful tasks. In the TELUS context, waiting acts primarily as a categorical risk signal rather than a gradual quantitative effect.

8.5 Underestimation and Task Failure

RQ3: To what extent does underestimation contribute to task failure

Approach

We define underestimation using a one-size correction rule. Let s_i denote the estimated size of task i, LT_i its observed lead time, and $\text{RLT}(s)$ the reference lead time associated with size s. Let $\text{next}(s_i)$ denote the immediately larger size on the estimation scale.

A task is considered *underestimated* if it fails at its assigned size but would have succeeded if it had been estimated one size larger:

$$U_i = \begin{cases} 1, & \text{if } \text{LT}_i > \text{RLT}(s_i) \ \wedge \ \text{LT}_i \leq \text{RLT}(\text{next}(s_i)), \\ 0, & \text{otherwise.} \end{cases} \tag{8.8}$$

This definition captures the smallest upward correction that would change a task's outcome from failure to success. Tasks already assigned the largest size category (XXL) cannot be classified as underestimated because no larger category exists.

Results

Table 8.5 Underestimation per size under both indicators (RLT^{MED}, RLT^{SLE})

Size	%fail^{MED}	%fail^{SLE}	%U$^{MED}_{tot}$	%U$^{SLE}_{tot}$	%U$^{MED}_{fail}$	%U$^{SLE}_{fail}$	%fail$^{MED}_{cf}$	%fail$^{SLE}_{cf}$	Δ^{MED}	Δ^{SLE}
XS	42.95	15.01	10.35	0.00	24.11	0.00	32.59	15.01	10.35	0.00
S	37.50	15.00	5.78	4.22	15.42	28.12	31.72	10.78	5.78	4.22
M	40.33	15.02	12.09	5.14	29.97	34.25	28.24	9.88	12.09	5.14
L	38.60	15.04	7.02	2.88	18.18	19.17	31.58	12.16	7.02	2.88
XL	40.00	15.09	14.72	10.19	36.79	67.50	25.28	4.91	14.72	10.19
XXL	39.13	15.94	0.00	0.00	0.00	0.00	39.13	15.94	0.00	0.00
ALL	**40.96**	**15.03**	**10.08**	**2.68**	**24.62**	**17.86**	**30.88**	**12.34**	**10.08**	**2.69**

Legend. %fail*: baseline failure rate at the task's own size. %U$^{\star}_{tot}$: % of all tasks at this size that are underestimated. %U$^{\star}_{fail}$: % of failed tasks recoverable by one-size correction. %fail$^{\star}_{cf}$: counterfactual failure rate if all tasks were sized one notch larger. $\Delta^{\star}$: drop in failure rate under the counterfactual. XXL has no larger size, so its underestimation is undefined.

Underestimation explains a meaningful share of task failure. Table 8.5 shows that 24.62% of failed tasks are recoverable by a one-size correction under RLT^{MED}, and 17.86% under RLT^{SLE}. A non-trivial fraction of failures is therefore associated with optimistic initial sizing rather than with execution alone. At the aggregate level, the counterfactual effect is substantial: if all tasks had been sized one category higher, the global failure rate would decrease from 40.96% to 30.88% under RLT^{MED}, and from 15.03% to 12.34% under RLT^{SLE}.

Underestimation is not uniformly distributed across sizes. The effect varies considerably across size categories. Under RLT^{MED}, the largest recoverable shares appear for XL (36.79% of failed tasks) and M (29.97%), followed by XS (24.11%) and L (18.18%). Under RLT^{SLE}, XL becomes a much stronger outlier, with 67.50% of failed XL tasks recoverable by a one-size correction, whereas XS drops to 0%. This contrast shows that the contribution of underestimation depends both on the size category and on the reference indicator used.

Underestimation is concentrated in middle and upper size transitions. Across both indicators, M and XL emerge as the most important categories. M shows a stable recoverable share under both thresholds (29.97% under RLT^{MED}; 34.25% under RLT^{SLE}), suggesting persistent calibration

Table 8.6 Significance of size-level variation in underestimation

Test	**Scope**	**MED** p	**SLE** p
χ^2 independence	all sizes	< 0.001	< 0.001
z-test	XS vs. S	0.003	< 0.001
z-test	S vs. M	< 0.001	0.268
z-test	M vs. L	< 0.001	0.002
z-test	L vs. XL	< 0.001	< 0.001

difficulty in this mid-range category. XL shows the strongest effect, especially under RLT^{SLE}, pointing to a marked calibration gap between XL and XXL. By contrast, XS does not behave as a systematically underestimated category: failed XS tasks under RLT^{SLE} are not recoverable by a one-size correction (0%), suggesting that these failures reflect stronger scope deviations or misclassification rather than mild underestimation.

Most failures remain unexplained by one-step underestimation alone. Although underestimation is meaningful, it does not explain the majority of failures. Even after a one-size correction, 75.38% of failed tasks under RLT^{MED} and 82.14% under RLT^{SLE} remain failed. This residual points to additional contributing factors, including execution-side interruptions such as waiting states (RQ2), stronger misclassification, or underestimation by more than one size category.

The variation in underestimation across sizes is statistically significant. Table 8.6 confirms that underestimation rates differ significantly across size categories under both indicators. Chi-square tests of independence reject the hypothesis of uniform underestimation rates at $p < 0.001$ under both RLT^{MED} and RLT^{SLE}. Pairwise z-tests between adjacent sizes show that under RLT^{MED}, every transition is significant, producing an alternating pattern in which recoverability rises from S to M, drops from M to L, and rises again from L to XL. Under RLT^{SLE}, the L→XL transition (19.17%→67.50%, $p < 0.001$) emerges as the largest single contrast in the analysis and identifies the L/XL boundary as the principal calibration gap in this environment.

Overall, RQ3 shows that underestimation is a meaningful but partial source of task failure. Its effect is strongest in the middle and upper parts of the size scale, especially at the L/XL boundary, rather than being concentrated uniformly among the smallest tasks. In the TELUS

context, targeted calibration at specific adjacent-size transitions appears more promising than treating estimation problems as a uniform tendency across all sizes.

8.6 Multitasking, Workload, and Task Failure

RQ4: What is the impact of multitasking and workload distribution on task failure

Approach

We define two related metrics: *workload* and *multitasking value* (MV).

Let d denote a developer and t a time instant. The workload of developer d at time t is:

$$\mathrm{WL}_d(t) = \sum_{i \in \mathcal{A}_d(t)} w(i), \tag{8.9}$$

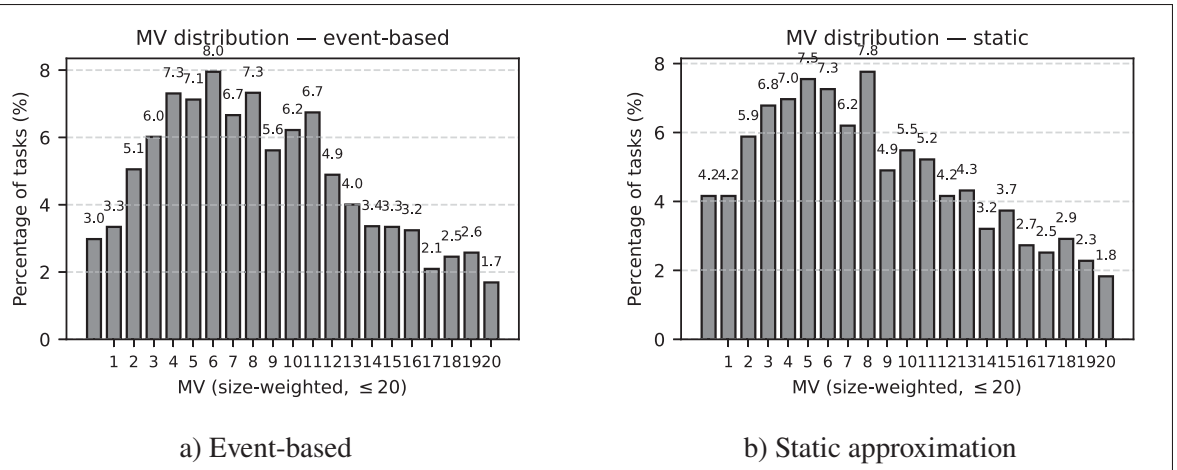
where $\mathcal{A}_d(t)$ is the set of active tasks assigned to d, and $w(i)$ is the Fibonacci-mapped weight of task i. This size-weighted formulation distinguishes between parallel work on small versus large tasks.

For a task i assigned to developer d, we define:

$$\mathrm{MV}_i = \frac{1}{|\mathcal{T}_i|} \sum_{t \in \mathcal{T}_i} \mathrm{WL}_d(t), \tag{8.10}$$

where $\mathcal{T}_i$ is the time interval during which task i is active.

We compute MV using two strategies. The *event-based* method reconstructs workload from detailed task events, whereas the *static* method estimates workload from temporal overlap under fixed assignment assumptions. We then compare MV across failed and successful tasks and examine whether failure rates vary with MV under both RLT^{MED} and RLT^{SLE}.



a) Event-based b) Static approximation

Figure 8.3 Distribution of size-weighted MV under both computation methods

Results

Multitasking levels are moderate under both computation methods. Figure 8.3 shows that MV is centered in the mid range under both approaches, with median values between 6 and 9 size-weighted units. The event-based method produces a wider distribution, including some values above 20, whereas the static approximation yields a narrower and more stable distribution. This difference suggests that some extreme values in the event-based version reflect short-lived overlaps captured by fine-grained events. Both methods indicate a moderate level of parallel work, consistent with sustainable Kanban practice.

Failed and successful tasks show very similar multitasking levels. Figure 8.4 compares mean and median MV between failed and successful tasks under both reference indicators and both computation methods. In all four configurations, the differences are small (at most one to two MV units). Moreover, the direction is not stable: under the event-based method, failed tasks have slightly lower MV than successful tasks, whereas under the static method the opposite pattern appears in one condition. This lack of consistency is itself informative: a systematic multitasking–failure relationship would produce a stable direction across operationalizations.

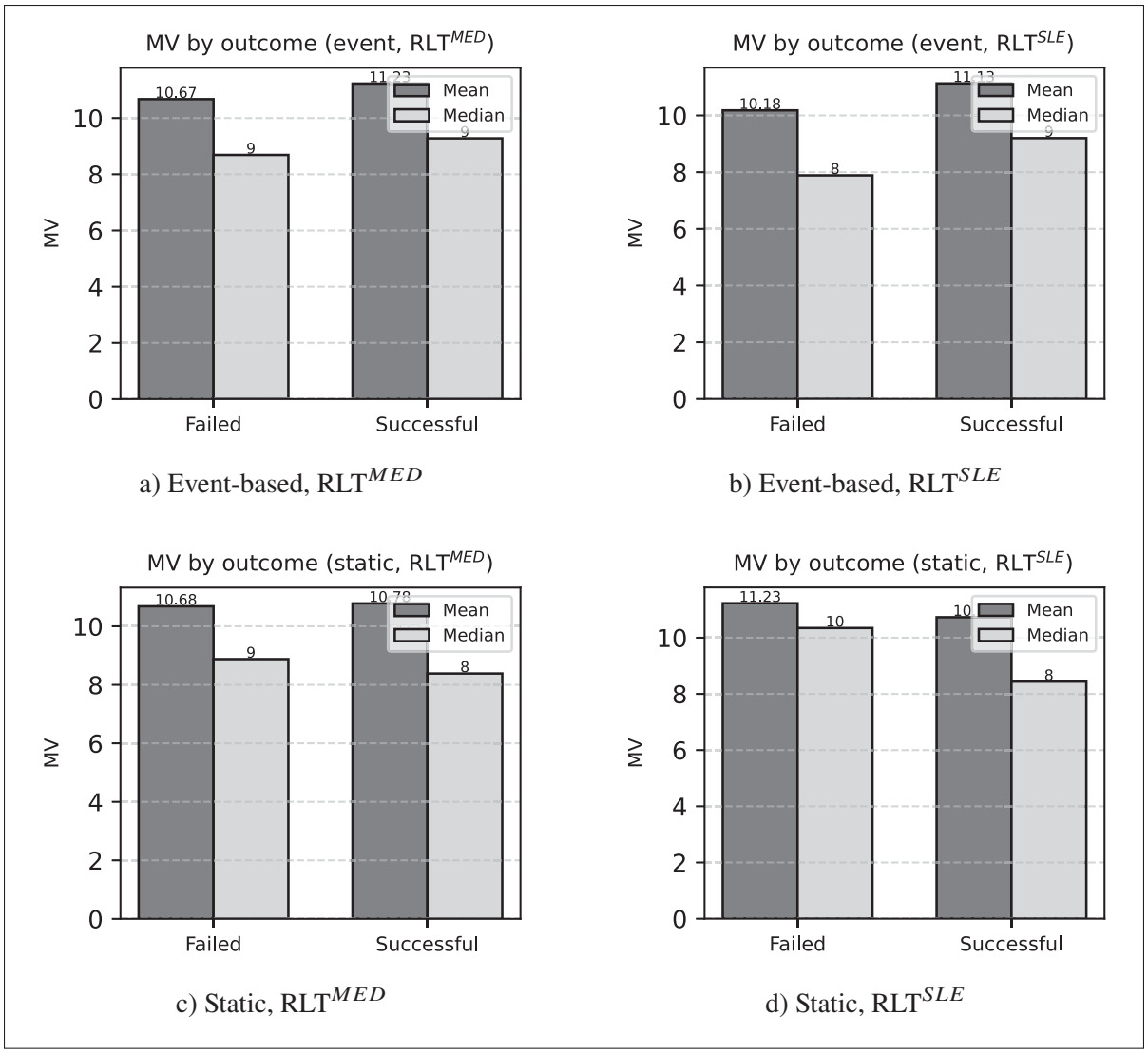


Figure 8.4 Mean and median MV by task outcome, under both computation methods and both reference indicators

Failure rates do not increase with multitasking intensity. Figure 8.5 shows that failure rates remain broadly flat across MV levels under both computation methods and both indicators. Under RLT^{MED}, rates fluctuate around the 35%–45% range (event-based) and 25%–40% range (static) without any monotonic trend. Under RLT^{SLE}, they remain close to the structural baseline of approximately 10%–15%. Local peaks appear at high MV values, but they are inconsistent across methods and likely reflect sparse data in the tail of the distribution rather than a systematic effect.

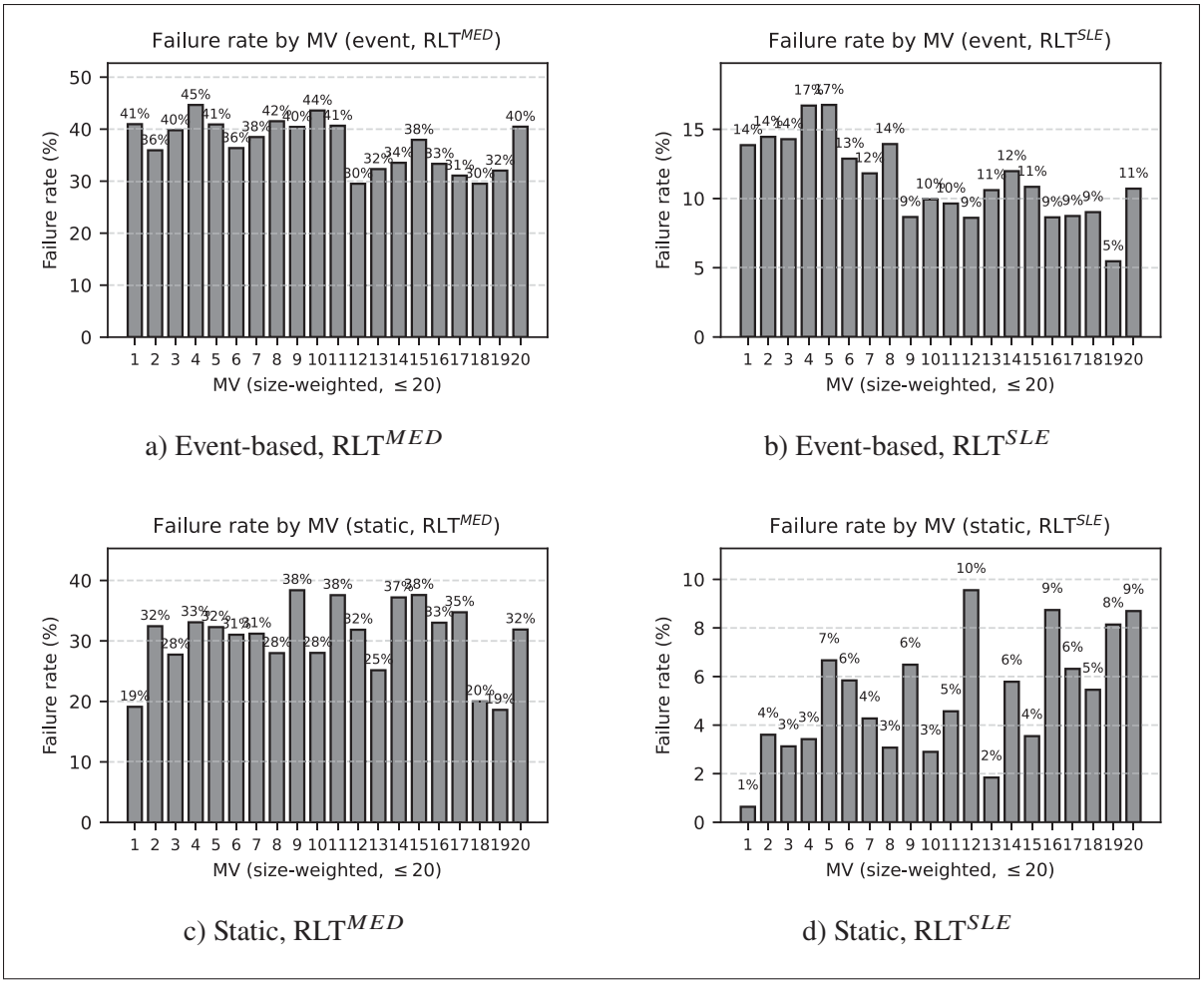


Figure 8.5 Failure rate per integer MV bin (capped at 20 for readability), under both computation methods and both reference indicators

Statistical tests confirm the absence of a practically meaningful effect. Table 8.7 reports Mann–Whitney U and Spearman rank correlation results under each method–indicator combination. Three of four configurations reach statistical significance at $\alpha = 0.05$, but two observations render this significance practically uninformative. First, all Spearman correlations are negligible in magnitude ($|\rho| \leq 0.05$), explaining less than 0.2% of the variance in task outcome. Second, and more importantly, the direction of the effect is inconsistent across methods: event-based results yield negative correlations (failed tasks have slightly lower MV), while the static SLE case yields a positive correlation. The statistical significance observed in some cells reflects the large sample size ($n > 6{,}000$) rather than a coherent multitasking–failure relationship.

Table 8.7 Association between MV and task failure. $|\rho|$ values are negligible and directionally inconsistent

Method	Indicator	Mann–Whitney p	Spearman ρ (p)
event	MED	0.012	−0.03 (0.012)
	SLE	0.0008	−0.05 (0.0008)
static	MED	0.386	+0.01 (0.386)
	SLE	0.007	+0.04 (0.007)

Overall, RQ4 shows that multitasking is not a meaningful driver of task failure in the TELUS context. Even under a size-weighted definition that captures the intensity of concurrent work, tasks executed at higher MV levels do not fail more often in any consistent way.

8.7 Prediction of Task Failure

RQ5: To what extent can issue delivery failure be predicted at task creation

Approach

We formulate this task as a binary classification problem that predicts, at task creation time, whether a newly created task will later be labeled as a failure under the reference indicators defined in Section 8.2.4. To prevent leakage, we use only point-in-time information available at creation.

Metric collection

We extract the point-in-time features summarized in Table 8.8. These features capture task properties, temporal context, and prior author-, assignee-, and project-level history. All history-based variables are computed strictly from events that occurred before the task was created.

Data preparation

We represent textual fields using TF–IDF (for ML models) . To reduce the influence of extreme values, we remove lead-time outliers above the 99th percentile before labeling outcomes. Because the target classes are imbalanced, we apply SMOTE on the training data only, while keeping the test data unchanged.

Model evaluation

We evaluate two model families. The first consists of conventional ML models: Decision Tree, Random Forest, and Gradient Boosting. These models provide a compact comparison between single-tree learning, bagging-based ensembles, and boosting-based ensembles. We train and evaluate them using an 80/20 train–test split.

The second family consists of locally hosted LLMs, namely `llama3.2:1b`, `qwen2.5:1.5b`, `gemma2:2b`, and `gemma4:E4B`. We use local models to satisfy the security and confidentiality requirements of the TELUS environment, and we focus on small models to keep inference practical in an industrial setting. For the LLM-based evaluation, we consider both one-shot prompting and few-shot prompting. In the few-shot setting, each prompt includes five success examples and five failure examples.

To control computational cost, we repeat each evaluation over 10 bootstrap runs and report median performance. This setup enables a comparison between standard predictive models and locally deployable LLM-based alternatives under a realistic industrial constraint: failure risk must be estimated at task creation using only information available at that moment. For all models we compute precision, recall, F1, and feature importance rankings using sk-esd.

Results

Random Forest and `gemma2:2b` with few-shot prompting are the strongest predictors. As shown in Table 8.9, Random Forest achieves the best AUC among supervised classifiers (0.570 under

Table 8.8 Point-in-time features available at task creation. All history-based features are computed strictly from events prior to task creation to prevent leakage

Feature	Description	Feature	Description
size	T-shirt size mapped to Fibonacci weight.	title	Free-text task title.
description	Free-text task description.	labels	GitLab labels attached to the task.
title_len	Number of characters in the title.	desc_len	Number of characters in the description.
has_description	1 if the description is non-trivial.		
month	Month of task creation.	day_of_week	Day of week of task creation.
hour	Hour of day of task creation.		
author_n_tasks	Number of previously authored tasks.	author_fail_rate$^{\text{MED}}$	Prior failure rate under RLT^{MED}.
author_fail_rate$^{\text{SLE}}$	Prior failure rate under RLT^{SLE}.	author_underest_rate$^{\text{MED}}$	Prior underestimation rate under RLT^{MED}.
author_underest_rate$^{\text{SLE}}$	Prior underestimation rate under RLT^{SLE}.		
assignee_n_tasks	Number of previously assigned tasks.	assignee_fail_rate$^{\text{MED}}$	Prior failure rate under RLT^{MED}.
assignee_fail_rate$^{\text{SLE}}$	Prior failure rate under RLT^{SLE}.	assignee_underest_rate$^{\text{MED}}$	Prior underestimation rate under RLT^{MED}.
assignee_underest_rate$^{\text{SLE}}$	Prior underestimation rate under RLT^{SLE}.	assignee_avg_lt_hours	Average lead time over prior tasks.
author_in_flight	Author's open tasks at creation.	assignee_in_flight	Assignee's open tasks at creation.
project_in_flight	Project's open tasks at creation.		
project_n_tasks	Number of previously closed tasks in the project.	project_fail_rate$^{\text{MED}}$	Project-level failure rate under RLT^{MED}.
project_fail_rate$^{\text{SLE}}$	Project-level failure rate under RLT^{SLE}.		

RLT^{MED}), followed by Gradient Boosting (0.562). Among local LLMs, `gemma2:2b` with few-shot prompting delivers the strongest overall performance, reaching AUC = 0.530 under RLT^{MED} and AUC = 0.574 under RLT^{SLE}—the highest value across all configurations. Supervised methods are strongest under RLT^{MED}, while `gemma2:2b` leads under RLT^{SLE}. Notably, `gemma4:E4B`, a more recent reasoning-capable model, performs comparably to `gemma2:2b` despite its larger parameter budget and thinking-before-answering design, suggesting that reasoning-style inference provides no clear advantage over standard instruction tuning on this classification task.

Author history and description features dominate the supervised signal. Feature-importance analysis of Random Forest shows that the strongest predictors are author-side estimation-history features—`author_n_tasks`, `author_fail_rate`, and `author_underest_rate`—together with description-related features and assignee lead-time history (`assignee_avg_lt_hours`). This complements RQ3 by showing that the *history* of estimation bias, not just the size boundary itself, carries predictive signal at task creation time.

8.8 Connecting Task Estimation to Peer Review Effort

The analyses presented in this chapter examine size-based task estimation from an execution perspective. However, the broader objective of this thesis is to understand and optimize peer

Table 8.9 RQ5 prediction performance. Best AUC per indicator in bold

	RLT^{MED}				RLT^{SLE}			
Method	**P**	**R**	**F1**	**AUC**	**P**	**R**	**F1**	**AUC**
Supervised classifiers								
Decision Tree	.414	.357	.389	.510	.115	.295	.170	.479
Random Forest	.461	.294	.361	**.570**	.054	.007	.012	.510
Gradient Boosting	.455	.239	.313	.562	.058	.013	.022	.517
Local LLMs (1s = 1-shot, fs = few-shot)								
llama3.2:1b (1s)	.475	1.00	.644	.505	.154	1.00	.267	.506
llama3.2:1b (fs)	.473	1.00	.642	.500	.152	1.00	.264	.500
qwen2.5:1.5b (1s)	.486	.989	.652	.525	.158	1.00	.273	.522
qwen2.5:1.5b (fs)	.486	.977	.649	.525	.161	.929	.274	.528
gemma2:2b (1s)	.462	.276	.345	.494	.205	.286	.239	.544
gemma2:2b (fs)	.515	.391	.444	.530	.214	.429	.286	**.574**
gemma4:E4B (1s)	.484	.874	.623	.519	.161	.857	.271	.528
gemma4:E4B (fs)	.463	.724	.565	.486	.167	.786	.275	.540

review effort. It is therefore important to clarify how the estimation-related findings connect to the review stage. In practice, task estimation and peer review describe two connected views of the same software process. A task that is underestimated, interrupted, or poorly stabilized during execution may later reach review as an MR that requires more clarification, more revision, or longer reviewer involvement. Conversely, excessive review effort may reveal that part of the task complexity was not adequately captured at planning time.

To examine this connection, we complement the task-level analysis with a review-effort dimension extracted from the same Kanban workflow. Specifically, we compute the cumulative time each task spends in the *In Review* column, using the same interval reconstruction procedure applied to waiting states. We derive three review-related indicators: whether the task entered review, the cumulative number of hours spent in review, and the ratio between review time and total task lead time. These indicators do not replace the MR-based effort measures studied in the previous chapters, but they provide an execution-level view of how much of a task's lifecycle is consumed by review.

Across the 13,419 tasks analyzed in this chapter, 1,319 tasks (9.8%) spent validated time in the *In Review* column. This indicates that review is not an explicit workflow stage for all tasks, at least as observed through the reconstructed Kanban traces. This observation is important for the thesis because it shows that review effort is not only a question of how much effort reviewed

tasks require, but also of which tasks enter review in the first place. It therefore motivates the next chapter, which examines self-managed MRs and the conditions under which review decisions may be adapted.

Among reviewed tasks, the median review duration is 55 hours, with the 90th percentile reaching approximately 490 hours. Review exposure is not uniformly distributed across estimated sizes. It is highest for M and L tasks, where 23.4% and 22.7% of tasks enter review, respectively, and lower at both ends of the size spectrum. However, when tasks are reviewed, median review duration generally increases with size, ranging from 48.6 hours for XS tasks to 124 hours for XXL tasks. This suggests that review exposure and review duration capture different aspects of review burden: medium and large tasks are more likely to enter review, while larger tasks tend to require longer review once they do.

The relationship between review exposure and task failure further supports the link between upstream execution conditions and downstream review burden. Under the median-based reference indicator (RLT^{MED}), tasks that enter the review column fail more often than tasks that do not enter review, with failure rates of 47.3% and 39.4%, respectively. This corresponds to a relative risk of 1.20 and is statistically significant ($\chi^2 = 27.14$, $p < 0.001$). Under the SLE-based reference indicator (RLT^{SLE}), the association is weaker and not conventionally significant. At the size level, the review–failure association is strongest for XS, S, and L tasks under RLT^{MED}, suggesting that review exposure becomes especially informative when tasks are expected to flow quickly but instead require additional review-stage processing.

A second result distinguishes review from waiting states. In the main analysis, waiting exposure is associated with task failure, but waiting duration itself does not clearly distinguish failed from successful tasks. Review behaves differently. Among reviewed tasks, failed tasks spend significantly more time in review than successful tasks under RLT^{MED}, with median review durations of 74.7 hours and 48.0 hours, respectively ($p = 5.2 \times 10^{-6}$). This indicates that review duration carries a quantitative signal of process burden, rather than only indicating whether a task passed through a specific workflow state. In other words, longer review time may reflect

active iteration, rework, clarification, or re-review, whereas waiting states more often represent the presence of an external dependency.

The strongest bridge between estimation and review effort appears in the analysis of underestimated tasks. Among failed reviewed tasks, underestimated tasks spend a substantially larger share of their lead time in review than other failed tasks. Under RLT^{MED}, the median review ratio is 0.155 for underestimated tasks, compared with 0.046 for other failed tasks, representing a 3.4-fold difference ($p = 2.3 \times 10^{-7}$). This result suggests that underestimation is not only associated with task failure in general; it also materializes as disproportionate review burden. In practical terms, tasks that are too small relative to their actual execution complexity may shift part of the unresolved complexity to the review stage.

Finally, we assess whether high-review-effort tasks can be predicted at creation time using the same feature set used for task-failure prediction. We define high-review-effort tasks as those in the top 10% of review hours among reviewed tasks of the same size and train a Random Forest classifier using a chronological 80/20 split. The resulting performance is weak (AUC = 0.28, F1 = 0), partly because the positive class is rare. This result is nevertheless informative. It suggests that high review burden is difficult to identify from creation-time metadata alone and that useful early-warning systems may need to incorporate execution-time signals, such as waiting transitions, review entries, partial review duration, or repeated review-stage movements.

Overall, this bridge analysis strengthens the thesis argument that peer review effort should be studied as part of a broader DevOps workflow. Upstream estimation and execution conditions do not merely affect task delivery outcomes; they may also shape how much burden later appears in the review stage. The results suggest that review-effort optimization should not be limited to reviewer assignment, review rules, or MR-level process changes. It should also consider whether tasks are appropriately estimated, whether execution instability is detected early, and whether underestimated tasks are likely to transfer unresolved work into peer review.

8.9 Discussion

Overall, this chapter shows that upstream planning and execution conditions provide an important context for understanding peer review effort. The main implication is not that estimation failure, waiting states, or multitasking directly determine review effort in a simple way. Rather, these factors help explain how part of the burden later observed during review may originate before the MR is created. This supports the thesis argument that peer review effort should be analyzed as part of a broader DevOps workflow, rather than as an isolated review-stage phenomenon.

Estimation quality matters more than nominal task size. One important lesson from the analysis is that task size alone does not explain unsuccessful task outcomes. What matters is whether the assigned size is well calibrated with respect to the task's actual execution complexity. This distinction is important for peer review effort because an underestimated task may reach review with unresolved complexity that was not anticipated at planning time. In such cases, review can become the stage where hidden complexity is discovered, discussed, and corrected. Peer review effort may therefore act partly as a compensation mechanism for earlier estimation inaccuracies.

Execution interruptions should be interpreted as context signals, not direct causes. Waiting states provide useful information about execution instability, but they should not be interpreted mechanically as direct causes of failure or review burden. The results suggest that entering a waiting state is more informative than the duration of waiting itself. This indicates that waiting exposure captures the presence of dependencies, blockers, or coordination issues, while waiting duration may also reflect routine operational constraints that teams are able to absorb. For peer review analytics, this distinction matters because not every delay in the workflow should be treated as a sign of increased review effort. Some delays reflect passive waiting, whereas others indicate unresolved work that may later reappear during review.

Review differs from passive workflow delay. The bridge analysis helps clarify the specific role of review within the execution workflow. Unlike waiting states, review duration is more directly connected to active process burden. Time spent in review may involve clarification, rework,

re-review, or coordination around the submitted change. This supports the thesis position that review effort is not only a temporal delay, but a form of work embedded in the broader software process. At the same time, the fact that only a subset of tasks explicitly enters the review column shows that review effort also depends on review-routing decisions: some tasks undergo explicit review, while others may follow lighter or self-managed paths. This observation motivates the next chapter, which examines review decisions and the conditions under which review effort may be adapted.

Underestimation provides the strongest bridge between planning and review burden. The most important thesis-level implication is that underestimation appears to transfer part of the unresolved task complexity into the review stage. When a task is scoped too optimistically, implementation may proceed under assumptions that do not match the actual work required. The resulting MR may then require more discussion, correction, or stabilization before acceptance. This does not mean that underestimation is the only source of review effort, but it shows that review burden can be partly generated upstream. Consequently, optimizing peer review effort requires attention to task scoping and estimation calibration, not only to reviewer behavior or MR-level practices.

Early prediction requires execution-time monitoring. The weak creation-time prediction results suggest that downstream execution and review difficulties cannot be reliably anticipated from initial task metadata alone. This has an important practical implication: process-support tools should not rely only on static information available at task creation. Instead, they should update their assessment as the task evolves, incorporating execution-time signals such as waiting transitions, review entry, repeated review movements, and partial review duration. This supports a dynamic view of DevOps analytics, where risk and effort are monitored progressively rather than predicted once at the beginning.

Implications for peer review effort optimization. Taken together, the findings extend the thesis from measuring review effort to understanding where avoidable review burden may originate. Peer review effort is not only shaped by what happens during review; it is also influenced by how

work is planned, scoped, and stabilized before review begins. Therefore, reducing unnecessary review burden may require interventions earlier in the workflow, such as improving estimation calibration, detecting unstable tasks during execution, and identifying tasks whose complexity is likely to exceed their original scope. This reinforces the broader argument of the thesis: peer review effort optimization is a process-level problem, not only a review-stage problem.

8.10 Implications

8.10.1 Implications for our industrial partners

For our industrial partners, the results indicate that task outcome predictability depends less on fine-tuning workflow phase durations and more on improving estimation calibration. In particular, systematic underestimation should be treated as an upstream process signal, because underestimated tasks may later generate execution instability, additional coordination, and increased review burden.

This implication becomes especially important in the era of LLM-assisted development. As LLMs increasingly support coding, documentation, testing, and review preparation, organizations may be tempted to reduce estimated effort because part of the production work is automated. However, AI assistance does not eliminate human effort. Developers still need to understand the task, validate generated outputs, adapt them to the project context, ensure correctness, respond to review feedback, and take responsibility for integration decisions. Estimation practices should therefore evolve to distinguish between work accelerated by automation and the remaining human effort required for judgment, coordination, validation, and accountability. Otherwise, LLM-assisted work may be systematically underestimated, increasing the risk that hidden complexity is transferred downstream into peer review.

Industrial teams can operationalize this insight by monitoring estimation failure rates by size category, introducing lightweight re-estimation triggers when tasks exceed the reference lead time of their size bucket, and integrating estimation-quality indicators into retrospectives. These

practices would help teams adjust planning assumptions before they propagate into downstream rework and peer review effort, thereby supporting more realistic workload management and better process control.

8.10.2 Implications for researchers

For researchers, this chapter broadens the analytical boundary of peer review effort studies. It shows that part of the burden observed during review may originate from upstream planning and estimation conditions, rather than from the review stage alone. In particular, the role of underestimation suggests that estimation quality should be treated as a first-class variable in empirical models of peer review, rework, and delivery effort. Studies that ignore planning-side signals may over-attribute downstream rework to implementation or review factors, while part of the observed effort may actually reflect earlier scoping and estimation decisions.

This implication is methodological as well as conceptual. Future studies should, when possible, link issue-tracking data with MR traces to distinguish effort generated during review from effort inherited from upstream planning. Variables such as estimation accuracy, re-estimation events, waiting-state exposure, and workload indicators can help explain why some reviewed changes require more clarification, revision, or coordination than others. Such integration would support a more process-oriented view of peer review effort, where review is analyzed as one stage in an end-to-end delivery workflow rather than as an isolated event.

The chapter also provides reusable measurement choices for replication and extension. The reference lead time framework, using both median-based and service-level-expectation anchors, offers an operational way to define size-specific execution expectations from trace data. The underestimation indicator provides a practical method for identifying tasks whose assigned size was likely too optimistic, without relying on retrospective interviews. Similarly, waiting exposure, waiting share of lead time, and size-weighted multitasking can be extracted from Kanban-style issue traces in other organizations. Future work can reuse these indicators to test

whether the relationships observed in this industrial context generalize to other teams, domains, and workflow models.

8.11 Threats to Validity

This chapter is subject to several threats to validity. First, task outcome is operationalized using size-specific Reference Lead Time (RLT), which captures whether a task exceeds the delivery expectations associated with its estimated size. Although this definition is appropriate for studying estimation reliability in a continuous-flow industrial setting, it does not cover all possible dimensions of task success, such as business value, implementation quality, or downstream impact. As a result, the notion of failure used in this chapter should be interpreted as a deviation from expected execution duration rather than as a complete measure of project success.

Second, the analyses in this chapter are observational. Although they rely on a large industrial dataset and consistent metric definitions, they do not control for all potentially relevant confounding factors, such as developer experience, task-specific technical complexity, or differences in team practices across projects. Consequently, the reported associations should not be interpreted as direct causal effects. In particular, while the chapter argues that upstream planning conditions may shape later peer review effort, this influence is inferred from process relationships rather than measured directly within the same analytical model.

Third, the study is conducted within a single large industrial organization using GitLab-based Kanban workflows and TELUS-specific estimation practices. This may limit the generalizability of the findings to organizations that use different planning approaches, delivery processes, or estimation schemes. For example, teams working in Scrum-based environments or relying on different effort scales may exhibit different estimation dynamics and execution patterns.

Finally, although the analyses are based on deterministic scripts and documented procedures, replication depends on access to fine-grained industrial issue and event data. Such data are rarely available in public repositories with the same level of completeness, especially for waiting states,

assignment history, and estimation updates. This limits the direct reproducibility of the study, even though the analytical approach itself can be transferred to comparable industrial contexts.

8.12 Conclusion

This chapter examined task estimation and execution behavior in an industrial Kanban-based DevOps environment at TELUS to understand how upstream planning conditions may shape later process outcomes and peer review effort. The results show that task failure is frequent and relatively stable across estimated sizes and over time, suggesting that nominal task size alone does not explain unsuccessful outcomes. Instead, the findings highlight the importance of estimation calibration and execution-time signals. Waiting exposure is associated with higher failure rates, but waiting duration itself does not clearly distinguish failed from successful tasks, indicating that the presence of an interruption is more informative than its length. Multitasking, by contrast, shows no practically meaningful relationship with task failure in the studied setting.

The chapter also shows that underestimation explains a meaningful share of unsuccessful outcomes, ranging from 17.9% to 24.6% depending on the reference indicator. This result suggests that task failure is partly rooted in overly optimistic planning assumptions rather than only in execution difficulties. The bridge analysis further connects this finding to peer review effort: tasks that enter the *In Review* column represent a minority of the dataset, but reviewed tasks can require substantial time, and underestimated failed tasks spend a disproportionately larger share of their lead time in review. This indicates that unresolved complexity introduced upstream may later materialize as review burden.

From the perspective of this thesis, these findings are important because they show that peer review effort should not be interpreted only as a consequence of review-stage interactions. Part of the effort observed during review may originate before the MR is created, through inaccurate estimation, unstable execution, or insufficiently calibrated planning assumptions. Peer review may therefore act partly as a downstream correction mechanism, where hidden complexity is discovered, clarified, and resolved before integration.

Overall, this chapter extends the thesis's process-oriented view of peer review effort by linking review burden to upstream planning and execution conditions. It shows that optimizing peer review effort requires more than improving review practices alone. It also requires attention to how tasks are scoped, estimated, monitored, and re-evaluated during execution, so that avoidable complexity is not transferred downstream into peer review.

CHAPTER 9

TOWARDS RISK-AWARE PEER REVIEW EFFORT REDUCTION IN INDUSTRIAL DEVOPS

Samah Kansab[1] , Moragne Gillette[1] , Francis Bordeleau[1] , Ali Tizghadam[2]

[1] Department of Software Engineering and IT, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[2] TELUS, 25 York St, Toronto, Ontario, Canada M5J 2V5

Paper to be Submitted in the 49nd IEEE International Conference on Software Engineering on July 2026

Preamble

This chapter studies how peer review decisions relate to downstream CI/CD delivery reliability, with particular attention to reduced collaborative review paths that can be revealed through self-managed decisions such as self-approval. Across TELUS and Kaloom projects, the results show that self-approval is not uniformly associated with pipeline failure. Instead, contributor history, change characteristics, and project-specific safeguards provide stronger and more stable signals of downstream risk. Building on this observation, the chapter shows that information available at MR creation time can be used to identify low-risk candidates for reduced collaborative review or self-approval with useful predictive performance. The chapter also introduces a prototype web extension that provides live recommendations during MR inspection, illustrating how the proposed model can support risk-aware peer review allocation in practice. Overall, the chapter extends the thesis from upstream planning and MR-level effort analysis to downstream delivery reliability, supporting the thesis argument that peer review effort should be calibrated rather than uniformly minimized.

This chapter is adapted from a paper submitted at ICSME 2026 with a planned journal extension. The thesis version revises the introduction, conclusion, and overall framing to align the chapter with the objectives of this thesis.

9.1 Introduction

The previous chapters showed that peer review effort is not uniform across industrial DevOps workflows. Some MRs receive substantial collaborative review, whereas others follow lighter, self-managed, or more direct integration paths. This raises an important question for the optimization objective of this thesis: can peer review effort be reduced for some merge requests without compromising downstream delivery reliability?

This question is central because peer review is both valuable and costly. It supports defect detection, maintainability, readability, and knowledge sharing (Bacchelli & Bird, 2013; Bavota & Russo, 2015; McIntosh, Kamei, Adams & Hassan, 2016b), but it also consumes time, creates waiting, and introduces coordination overhead in fast-paced DevOps environments (Olewicki *et al.*, 2024b; Chen *et al.*, 2022). Optimizing peer review effort therefore does not mean reducing review indiscriminately. Rather, it requires identifying when collaborative review is necessary, when lighter review paths may be acceptable, and when reduced review may expose the delivery pipeline to unnecessary risk.

This problem must be studied from a downstream perspective. Because MRs sit immediately before integration and CI/CD execution, review decisions may influence delivery reliability. Reduced collaborative review may save effort for low-risk and well-prepared changes, but it may also be harmful if insufficiently examined changes increase the likelihood of pipeline failure. Prior work has shown that review outcomes interact with CI build outcomes, and that review-related factors such as discussion and review duration remain relevant for downstream integration behavior (Zampetti, Bavota, Canfora & Di Penta, 2019). In this chapter, we therefore examine the relationship between peer review decisions and downstream CI/CD outcomes. Among the observable signals of lighter review, self-approval is particularly informative, since it indicates that a contributor played a central role in approving or integrating the change with limited external review involvement. However, the goal is not to promote systematic self-approval, but to determine whether some MRs that follow reduced-review paths can be identified as low-risk candidates based on downstream evidence.

The chapter adopts an inverse-risk perspective. Instead of only asking which review decisions are associated with pipeline failure, it also asks which MRs appear historically low risk despite receiving limited collaborative review. This perspective is consistent with the broader argument of the thesis: peer review effort optimization is a calibration problem, not a minimization problem. Effort should be reduced where collaborative review appears to add limited value, but preserved where the characteristics of the change, the contributor, or the project indicate higher downstream risk. This view is also consistent with prior work showing that merge-request acceptance and integration outcomes depend on a combination of review, contributor, and project factors. (Zhang, Yu, Gousios & Rastogi, 2023b; Saidani, Ouni, Chouchane & Mkaouer, 2020; Ghaleb, Yu & Briand, 2019; Cassee, Vasilescu & Serebrenik, 2020)

The objective of this chapter is therefore to investigate whether peer review effort can be adapted based on downstream delivery risk. More specifically, the chapter examines which MR and review characteristics are associated with CI/CD pipeline failure, whether low-risk candidates for reduced collaborative review or self-approval can be identified using information available at MR creation time, and which features drive such recommendations. Across the studied Kaloom and TELUS projects, we find that self-managed review decisions are not uniformly associated with poorer downstream outcomes. The relationship between self-approval and pipeline failure is generally weak and project-dependent. We also show that a creation-time classifier can identify low-risk candidates for reduced collaborative review under conservative historical conditions, and that the most influential signals behind such recommendations are change size, author reliability, and the richness of the submitted MR description. To illustrate the practical use of this approach, the chapter also introduces a prototype web extension that provides live recommendations during MR inspection.

The main contributions of this chapter are:

- an empirical analysis of the relationship between MR characteristics, peer review decisions, and downstream CI/CD pipeline failure across industrial projects from Kaloom and TELUS;
- an inverse-risk analysis that identifies review situations not strongly associated with pipeline failure and that may therefore represent candidates for reduced collaborative review;

- a creation-time recommendation model that identifies low-risk candidates for reduced collaborative review or self-approval under conservative historical conditions, supporting risk-aware peer review effort adaptation;
- an analysis of the features driving this recommendation, showing that low-risk recommendations are mainly governed by change complexity, contributor reliability, and MR description quality rather than by a single categorical review decision;
- a prototype web extension that provides live recommendations on whether an MR appears suitable for reduced collaborative review or self-approval, illustrating how the proposed model can be integrated into developers' existing review workflow to support risk-aware review-effort adaptation in practice.

9.2 Study Design

9.2.1 Research Questions

The chapter is guided by the following three research questions:

RQ1 **Which peer review and MR characteristics are associated with downstream CI/CD pipeline failure?** This question provides the explanatory foundation of the chapter. Before reasoning about whether peer review effort can be reduced for some MRs, it is first necessary to understand which review decisions, process conditions, and MR characteristics are associated with downstream failure. In particular, this question examines whether self-managed review practices, process bypass conditions, review engagement, change complexity, and author history are related to pipeline failure across projects.

RQ2 **How accurately can creation-time information be used to recommend low-risk candidates for reduced collaborative review or self-approval?** While RQ1 studies associations with downstream reliability, RQ2 moves toward proactive support. Its objective is to determine whether information available at MR creation time is sufficient to identify MRs whose historical counterparts received little or no meaningful review

contribution and experienced no pipeline failure. Such cases are treated as candidates for risk-aware review-effort adaptation rather than as evidence that review can always be bypassed.

RQ3 Which creation-time features have the strongest influence on the reduced-review recommendation? While RQ2 evaluates whether low-risk candidates can be predicted with useful accuracy, RQ3 focuses on the interpretability of that prediction. Its objective is to identify which creation-time features drive the recommendation most strongly, and thus to better understand what kinds of MRs are judged by the model as more suitable for reduced collaborative review or self-approval. More specifically, this question examines whether the recommendation is shaped primarily by the technical characteristics of the submitted change, the amount of context provided at submission, or the author's prior delivery and self-review history.

9.2.2 Data

The study combines both industrial datasets of the thesis: the five Kaloom technical groups described in Section 3.3.1 and the five TELUS TINAA repositories described in Section 3.3.2. All analyses are performed on merged MRs with at least one recorded pipeline.

9.2.3 Analysis Strategy

The analysis strategy varies according to the objective of each research question, but follows a common overall logic. First, we operationalize downstream delivery reliability through a binary pipeline-failure outcome derived from the status of the MR's recorded pipeline executions. This outcome serves as the main dependent variable for the explanatory analyses of the chapter. Second, we relate this outcome to several complementary dimensions of peer review and MR context, including self-review decisions, process bypass indicators, review engagement, change complexity, and author history.

For RQ1, we use descriptive and comparative analyses to examine which review-related and MR-related characteristics are associated with pipeline failure across projects. Binary indicators are analyzed using chi-square tests of independence together with odds ratios, whereas numeric metrics are analyzed using Mann–Whitney U tests and Cliff's δ effect sizes. These analyses provide a project-level view of whether reduced collaborative review is consistently associated with downstream failure, or whether other factors better explain delivery risk.

For RQ2, we shift from explanation to recommendation. We first derive a conservative annotation of whether a historical MR appears to be a low-risk candidate for reduced collaborative review based on review outcomes and pipeline success. We then train and evaluate classifiers using only creation-time features so that the resulting model can be applied when an MR is submitted, before any review activity has taken place.

For RQ3, we complement RQ2 with an analysis of feature importance based on the same creation-time features. This analysis identifies which dimensions of the MR and its author most strongly drive the recommendation and clarifies whether the model relies primarily on change complexity, contributor history, or submission context.

9.3 Results

9.3.1 Review Decisions and Factors Associated with Downstream Pipeline Failure

RQ1: Which peer review and MR characteristics are associated with downstream CI/CD pipeline failure

Approach

To investigate the relationship between peer review decisions and downstream delivery reliability, we examine whether selected MR characteristics are associated with pipeline failure after integration. The analysis follows three steps. First, we define pipeline failure as the main dependent variable and restrict the dataset to merged MRs with at least one recorded pipeline,

so that each observation corresponds to an actual post-review delivery outcome. Second, we organize the explanatory metrics into five complementary categories that capture who controlled the review outcome, whether standard review safeguards were bypassed, how much review engagement the MR received, how complex the submitted change was, and what was known about the author's prior delivery history at the time of submission. Third, we apply statistical procedures suited to each variable type in order to compare failed and non-failed MRs across projects.

This organization allows the analysis to distinguish between several possible explanations of downstream risk. More specifically, it examines whether pipeline failure is more strongly associated with self-managed review behavior, reduced procedural control, low review engagement, change-related complexity, or the author's prior delivery reliability.

Metric Categories

Self-Review Decision Metrics

The first category captures whether the MR author participated directly in the approval or merge decision of the submitted change. These metrics are central to this chapter because they represent cases in which the usual separation between contributor and reviewer is weakened or absent. By measuring the extent to which review authority remained external or was instead retained by the author, these metrics allow us to examine whether self-managed review decisions are associated with downstream pipeline failure. More broadly, they support the thesis's objective of determining whether some forms of reduced peer review effort can be tolerated without compromising delivery reliability.

Process Bypass Indicators

The second category captures conditions under which an MR may have bypassed part of the expected review workflow. Whereas the previous category focuses on *who* made the decision,

this category focuses on *whether* the decision occurred under reduced procedural control. These indicators include merging without recorded approval, proceeding without any external reviewer, modifying CI/CD configuration files that directly affect the pipeline, and enabling auto-merge conditioned on pipeline success. Taken together, these metrics allow us to examine whether workflow shortcuts or automated safeguards are associated with different levels of downstream delivery risk. They are especially useful for distinguishing practices that may genuinely weaken process control from those that primarily reduce coordination overhead without measurable negative consequences.

Review Engagement Metrics

The third category captures the extent of human participation during the review process. Since downstream reliability may depend not only on formal approval but also on the depth of review interaction, we include several indicators of review engagement, namely the number of participating reviewers, approvals, comments, discussion threads, and the elapsed time to complete peer review. These metrics allow us to assess whether stronger review engagement is systematically associated with lower pipeline-failure risk, or whether lighter forms of review may already be sufficient in some contexts. In this way, they contribute to the broader question of how peer review effort can be allocated more efficiently while preserving adequate delivery safeguards.

Change Complexity Metrics

The fourth category characterizes the size and structure of the submitted change. These metrics include the number of commits, files changed, total churn, and the length of the MR description. They are included primarily as contextual variables, since downstream pipeline failure may be influenced by the intrinsic complexity of the change rather than by review decisions alone. Accounting for this dimension is therefore necessary to separate associations linked to peer review practices from those more strongly related to the technical scope or documentation quality

of the MR. In addition, description length provides a coarse proxy for the amount of contextual information made available at submission time.

Author History Metrics

The fifth category captures the contributor's prior record at the time of MR submission. These metrics include the cumulative number of prior MRs, the number of prior pipeline failures, and the resulting historical failure rate. All such measures are computed as rolling aggregates over prior MRs within the same project group so that only information available at the time of review is considered. This design avoids information leakage and ensures that the analysis remains temporally valid. Substantively, these metrics allow us to assess whether downstream delivery risk is better explained by the author's prior reliability than by the immediate review decision itself. If so, this would support a more risk-aware allocation of peer review effort, in which stricter review attention is directed toward changes with higher expected risk rather than being applied uniformly across all contributors.

Pipeline Outcome

The dependent variable in this RQ is *pipeline failure*, defined as a binary indicator of whether the MR's recorded pipeline outcome ended in a failed status. Pipeline status is extracted from the `head_pipeline` field in the MR metadata, which records the status of the latest pipeline associated with the MR. We restrict the analysis to merged MRs with at least one recorded pipeline, since pipeline outcomes are undefined for unmerged MRs or for MRs without pipeline executions. This restriction ensures that the dependent variable corresponds to an observed downstream delivery outcome.

Table 9.1 summarizes the metrics used in this RQ, organized by category.

Table 9.1 Summary of metrics used to study the relationship between peer review decisions and CI/CD pipeline outcomes

Category	Metric	Definition	Rationale
Self-Review Decisions	`is_self_approved`	The author's last approval action on the MR is an approval (extracted from system notes).	Identifies MRs where the author validated their own change.
	`is_self_merged`	The `merge_user` matches the MR author.	Identifies MRs where the author performed the integration.
	`is_self_managed`	The author approved, merged, and no other contributor approved the MR.	Captures the absence of any external review oversight.
	`is_self_assigned`	The MR assignee matches the author.	Indicates whether the review was explicitly delegated to the author.
Process Bypass	`merged_despite_failure`	The last pipeline had a failed status when the MR was merged.	Identifies integration despite a red pipeline.
	`merged_without_approval`	The MR was merged with zero recorded approvals.	Identifies integration without formal approval.
	`has_no_reviewer`	No non-author contributor participated through comments or discussions.	Flags MRs with zero external engagement.
	`touches_ci_config`	The MR modifies CI/CD configuration files.	Identifies changes that directly affect the pipeline definition.
	`merge_when_pl_succeeds`	The auto-merge feature was enabled.	Captures whether the pipeline was used as a merge gate.
Review Engagement	`participating_reviewers`	Count of distinct non-author contributors who left non-system comments.	Measures the breadth of reviewer involvement.
	`number_of_approvals`	Total approvals recorded on the MR.	Quantifies formal review sign-off.
	`human_comments`	Count of non-system notes on the MR.	Reflects the volume of reviewer feedback.
	`number_of_discussions`	Count of discussion threads.	Captures the number of review conversations.
	`review_duration_hours`	Hours between MR creation and merge.	Provides a temporal view of the review process.
Change Complexity	`total_commits`	Number of unique commits in the MR.	Reflects the granularity of the submitted change.
	`files_changed`	Number of files modified.	Measures the scope of the change.
	`mr_churn`	Sum of lines added and deleted.	Quantifies the volume of code modification.
	`description_length`	Character count of the MR description.	Proxies the amount of context provided at submission.
Author History	`author_prior_failure_rate`	Ratio of the author's prior pipeline failures to prior pipeline executions.	Captures the author's historical reliability with respect to CI/CD.
	`author_prior_pl_failures`	Count of the author's prior MRs with pipeline failures.	Provides an absolute measure of past failure frequency.
	`author_prior_mrs`	Count of the author's prior MRs in the project.	Contextualizes the failure rate (a 50% rate from 2 MRs differs from 100 MRs).

Statistical Analysis

To examine the relationship between each metric and pipeline failure, we apply complementary analytical procedures according to the type of variable considered.

For binary metrics, such as self-approval and absence of external reviewers, we use the chi-square test of independence to determine whether the presence of the characteristic is associated with pipeline failure. A result is considered statistically significant when the p-value is below 0.05. To complement statistical significance, we report Cramér's V as a measure of association strength and the odds ratio (OR) to quantify the direction and magnitude of the effect. An OR greater than one indicates that the presence of the characteristic is associated with higher failure risk, whereas an OR below one indicates lower failure risk.

For continuous and ordinal metrics, such as the number of commits and the author's prior failure rate, we use the Mann–Whitney U test to compare the distributions of failed and non-failed MRs. Because these variables are not assumed to follow a normal distribution, the Mann–Whitney test provides a robust non-parametric basis for group comparison. We report Cliff's δ as an effect-size measure and interpret it using the thresholds proposed by Romano et al. Romano, Kromrey, Coraggio, Skowronek & Devine (2006): negligible ($|\delta| < 0.147$), small ($|\delta| < 0.33$), medium ($|\delta| < 0.474$), and large ($|\delta| \geq 0.474$).

Taken together, these analyses provide an explanatory view of which review-related, contextual, and historical factors are most consistently associated with downstream CI/CD failure across the studied projects.

Results

Author's historical delivery reliability is more strongly associated with downstream pipeline failure than the immediate review decision itself. As shown in Table 9.4, the author's prior pipeline failure rate is significant in all studied projects and generally exhibits larger effect sizes than the self-review decision variables. This pattern is particularly clear in the TELUS projects, where the effect sizes reach medium to large magnitudes. A similar, although slightly less uniform, pattern is observed for the absolute number of prior pipeline failures. These results indicate that failed MRs are consistently associated with contributors who had a weaker prior delivery record. From the perspective of peer review effort optimization, this finding is

important because it suggests that downstream risk is more strongly tied to prior reliability than to any single review-decision flag considered in isolation.

Self-review decisions do not provide a stable indication of downstream failure across projects. Table 9.3 shows that self-approval is rarely significant, while the stricter self-managed condition provides little evidence of a consistent association with failure. Self-merging is significant in several projects, but its direction varies substantially: it is associated with higher failure risk in some projects, such as CP, DP, and CDAF, but with lower failure risk in others, such as MP, Platform, and Infrastructure. A similarly unstable pattern appears for self-assignment. Overall, these results do not support the view that self-managed review behavior is inherently unsafe. Rather, they suggest that its relationship with downstream failure is context-dependent and should be interpreted in light of project-specific practices and safeguards rather than through a uniform rule.

Process-control indicators provide a more nuanced picture. As shown in Table 9.3, enabling merge when the pipeline succeeds is associated with lower failure risk in several projects, with especially strong effects in the TELUS context. This suggests that pipeline-gated merging may act as a meaningful safeguard in some environments. By contrast, merging without recorded approval and proceeding without an external reviewer show mixed associations. They are linked to higher failure risk in some projects, but not in all, and may even appear protective in particular cases. This lack of stability indicates that reduced procedural control is not uniformly detrimental. Instead, its impact appears to depend on the broader process environment in which the MR is handled.

Review engagement shows some protective association with pipeline reliability, but this effect remains moderate and project-dependent. As shown in Table 9.4, failed MRs tend in several projects to involve fewer participating reviewers, fewer discussion threads, and fewer human comments, particularly in MP, NetApps, CDAF, and CNE-IP. Some of the strongest negative effects are observed for the number of discussion threads in projects CDAF and CNE-IP, suggesting that richer review interaction may help reduce downstream failure in certain contexts.

However, these patterns are not equally strong across all projects, and their effect sizes remain generally lower and less stable than those observed for author-history variables. Review engagement may therefore contribute to delivery reliability, but it does not by itself provide a universal explanation of downstream risk.

Change-related metrics also fail to show a uniform relationship with pipeline failure. Table 9.4 indicates that the number of commits, the number of changed files, and the description length vary in both significance and direction across projects. In several cases, failed MRs are associated with fewer commits or fewer changed files rather than larger changes. This suggests that change size alone is not a sufficient indicator of downstream delivery risk and that pipeline failure is shaped by project-specific conditions rather than by a simple complexity effect.

Time to complete peer review does not emerge as a meaningful indicator of downstream CI/CD reliability. As shown in Table 9.4, review duration is negligible in most projects and, when significant, does not follow a stable direction. In some projects, failed MRs are associated with slightly longer review duration, whereas in others they are associated with shorter duration, including project CDAF where the effect is large and negative. This result is consistent with the broader argument of the thesis: temporal measures alone provide only a partial view of process quality because they may reflect waiting time, urgency, or local coordination practices rather than the effectiveness of the review itself.

Cross-project heterogeneity is one of the most important characteristics of the results. Table 9.2 shows, for example, that project CNE-IP combines extremely high rates of self-merging, merging without approval, and absence of external reviewers with the lowest observed failure rate among all projects. By contrast, other projects with less extreme review configurations exhibit higher failure rates or more problematic associations between weaker review control and downstream outcomes. This heterogeneity suggests that low-collaboration review workflows are not universally detrimental. Their safety depends on the broader process environment in which they occur, including local engineering norms, pipeline constraints, and contributor experience.

Table 9.2 Prevalence of self-review decisions, process bypass indicators, and review engagement across the studied projects

Project	Self-Review (%)			Process Bypass (%)			Review Context (%)		MR Properties (Median)		
	Self-Appr.	Self-Mrgd.	Self-Mgd.	Desp. Fail.	W/O Appr.	Auto-Merge	No Rev.	Fail Rate	Commits	Files	Reviewers
CP	2.5	38.9	2.4	7.5	2.2	—	70.3	7.5	1	2	0
DP	2.6	23.9	2.2	11.0	0.2	—	47.4	11.1	1	3	1
MP	0.2	72.3	0.2	6.4	0.9	—	58.2	6.4	1	3	0
FPGA	0.4	36.0	0.0	12.9	1.5	—	55.3	12.9	2	3	0
PF	0.3	48.2	0.3	9.1	0.3	—	71.7	9.2	1	1	0
NetApps	9.5	87.2	9.0	11.4	40.7	—	52.1	11.4	2	2	0
Platform	18.9	63.6	17.3	14.1	44.0	—	61.5	15.4	3	3	0
Infrastructure	0.0	50.9	0.0	20.1	50.3	—	38.4	21.4	2	2	1
CDAF	6.0	63.8	5.9	16.5	56.8	—	56.9	16.5	3	3	0
CNE-IP	0.4	98.7	0.4	3.4	82.6	—	90.6	3.4	1	2	0

Table 9.3 Association between binary review flags and pipeline failure across all studied projects. Each cell reports the odds ratio (OR); boldface indicates statistical significance at $p < 0.05$ (chi-square test). OR > 1 indicates higher failure risk when the flag is true; OR < 1 indicates lower risk. Empty cells indicate insufficient data for the test

	Kaloom					TELUS				
Flag	**CP**	**DP**	**MP**	**FPGA**	**PF**	**NetApps**	**Platform**	**Infra.**	**CDAF**	**CNE-IP**
`is_self_approved`	1.04	1.16	3.20	1.33	1.29	1.30	1.15	—	0.89	8.88
`is_self_merged`	**1.29**	**1.30**	**0.64**	1.49	1.06	0.71	**0.61**	**0.33**	**4.46**	0.26
`is_self_managed`	0.93	1.18	3.20	—	1.75	1.23	1.17	—	0.92	8.88
`is_self_assigned`	1.53	**1.33**	0.63	0.80	1.37	**0.51**	**0.48**	1.04	**0.13**	0.23
`merged_despite_failure`	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
`merged_without_approval`	0.55	2.25	1.67	**7.03**	1.29	**1.69**	0.95	**0.34**	**4.35**	3.78
`has_no_reviewer`	1.11	**1.24**	**2.25**	0.84	0.87	**1.89**	1.23	**0.18**	**7.39**	1.86
`touches_ci_config`	**1.49**	**1.24**	1.17	**2.20**	0.95	**1.70**	**0.46**	1.56	**0.48**	1.04
`merge_when_pl_succ.`	**0.52**	**0.84**	0.91	1.13	1.29	**0.04**	**0.13**	**0.06**	**0.05**	0.87
`merged_w/_unresolved`	2.27	**3.89**	1.50	1.33	0.36	0.56	1.29	0.61	0.78	—

Note: ∞ denotes a tautological relationship (`merged_despite_failure` is a subset of the outcome variable). Boldface = $p < 0.05$. — = insufficient data (fewer than 5 observations in one group).

Taken together, the results suggest that peer review effort should not be optimized by systematically prohibiting self-managed review decisions. Instead, the evidence points toward a more selective approach in which reduced collaborative review may be acceptable for lower-risk MRs, while stronger external review is reserved for changes associated with higher delivery risk. In this respect, the strong role of author historical reliability, together with the potentially protective role of pipeline-gated merging, provides an empirical basis for moving toward risk-aware review policies rather than uniform governance rules.

Table 9.4 Cliff's δ effect sizes from Mann-Whitney U tests comparing numeric metrics between MRs with and without pipeline failure. Positive values indicate higher values in failed MRs; negative values indicate lower values. Boldface indicates statistical significance at $p < 0.05$. Effect size magnitudes: N = negligible ($|\delta| < 0.147$), S = small ($|\delta| < 0.33$), M = medium ($|\delta| < 0.474$), L = large ($|\delta| \geq 0.474$)

	Kaloom					TELUS				
Feature	**CP**	**DP**	**MP**	**FPGA**	**PF**	**NetApps**	**Platform**	**Infra.**	**CDAF**	**CNE-IP**
`author_prior_failure_rate`	**+.29**^{S}	**+.20**^{S}	**+.15**^{S}	**+.23**^{S}	**+.12**^{N}	**+.36**^{M}	**+.61**^{L}	**+.93**^{L}	**+.77**^{L}	**+.65**^{L}
`author_prior_pl_failures`	**+.33**^{M}	**+.16**^{S}	+.11^{N}	**+.24**^{S}	**+.15**^{S}	**+.30**^{S}	**+.43**^{M}	**+.71**^{L}	**+.65**^{L}	−.21^{S}
`participating_reviewers`	−.02^{N}	−.10^{N}	−.19^{S}	+.08^{N}	+.03^{N}	−.20^{S}	−.03^{N}	+.15^{S}	−.37^{M}	−.10^{N}
`number_of_discussions`	−.04^{N}	−.06^{N}	−.19^{S}	+.01^{N}	**+.07**^{N}	−.20^{S}	−.05^{N}	−.02^{N}	−.54^{L}	−.50^{L}
`human_comments`	−.03^{N}	−.02^{N}	−.20^{S}	+.01^{N}	+.01^{N}	−.12^{N}	−.03^{N}	−.05^{N}	−.11^{N}	−.01^{N}
`review_duration_hours`	**+.07**^{N}	−.02^{N}	−.10^{N}	+.09^{N}	+.06^{N}	−.15^{S}	−.05^{N}	+.17^{S}	−.56^{L}	−.15^{N}
`total_commits`	−.19^{S}	−.17^{S}	−.33^{M}	−.15^{S}	−.02^{N}	**+.12**^{N}	−.01^{N}	−.34^{M}	−.46^{M}	−.09^{N}
`files_changed`	−.04^{N}	−.05^{N}	−.11^{N}	+.12^{N}	−.02^{N}	**+.10**^{N}	+.00^{N}	−.59^{L}	−.08^{N}	+.08^{N}
`description_length`	−.10^{N}	**+.07**^{N}	−.09^{N}	−.06^{N}	+.01^{N}	−.23^{S}	**+.08**^{N}	**+.28**^{S}	**+.24**^{S}	−.01^{N}

9.3.2 Predicting When Reduced Collaborative Review Can Be Recommended Safely

RQ2: How accurately can creation-time information be used to recommend low-risk candidates for reduced collaborative review or self-approval

Approach

Following the findings of RQ1, we next examine whether it is possible to move from retrospective analysis to proactive support. More specifically, this research question investigates whether MRs for which external peer review would add little or no observable value can be identified as early as MR creation time. The objective is not to remove peer review indiscriminately, but to determine whether a recommendation model can distinguish lower-risk MRs, for which reduced collaborative review or self-approval may be acceptable, from MRs that still warrant external review.

The analysis follows three steps. First, we construct a conservative ground-truth label indicating whether a historical MR appears to be a low-risk candidate for reduced collaborative review, based on its observed review outcome and downstream pipeline result. Second, we train prediction models using only features available at MR creation time, so that the recommendation

Table 9.5 Metrics used to label historical MRs as *trivial*, *lightweight*, or *substantive* review. These metrics are derived from review outcomes and are used exclusively for annotation, not as model inputs

Metric	Definition	Justification
`has_pipeline_failure`	Whether any pipeline associated with the MR had a failed status.	A failed pipeline indicates that the change was not safely integrated; such MRs should not be considered self-approvable.
`human_comments`	Number of non-system comments left on the MR.	Zero comments from reviewers suggests that the review added little or no explicit feedback.
`number_of_discussions`	Number of discussion threads on the MR.	Few discussions indicate minimal review interaction beyond system activity.
`resolved_discussions`	Number of resolvable discussion threads that were resolved.	Resolved discussions indicate that the reviewer raised concerns requiring follow-up.
`post_submission_commits`	Number of commits added after MR creation.	Post-submission commits reflect rework that occurred during review.
`post_sub_churn`	Total lines added and deleted in post-submission commits (TELUS only).	Quantifies the volume of review-driven changes.

setting remains realistic and free of information leakage. Third, we evaluate the models through repeated out-of-bag validation in order to assess how reliably such low-risk candidates can be identified across projects.

Annotation Metrics

To construct the target label, we first collect the review-outcome metrics shown in Table 9.5. These metrics are used exclusively for annotation and are not used as model inputs.

Annotation of Reduced-Review Candidates

To train the recommendation model, we require a ground-truth label indicating which historical MRs appear to be plausible candidates for reduced collaborative review. Rather than relying on manual annotation, we derive this label automatically from the observed review outcomes recorded in the dataset. The underlying rationale is that if peer review produced no meaningful feedback, no substantial rework, and no pipeline failure, then the same MR may plausibly have been integrated through a lighter review path without changing the observed delivery outcome.

Each merged MR with at least one recorded pipeline is therefore classified into one of three review-outcome categories based on the metrics listed in Table 9.5:

Trivial review. The pipeline passed, the reviewer left zero human comments, no more than two discussion threads are present, no post-submission commits were added, and no discussions were resolved. In such cases, the review produced no observable feedback or requested change.

Lightweight review. The pipeline passed, the reviewer left at most one human comment, no more than three discussion threads are present, at most one post-submission commit was added, at most one discussion was resolved, and post-submission churn did not exceed 20 lines where this metric is available. In such cases, the review involved only limited interaction and at most minor rework.

Substantive review. The pipeline failed, or the review involved stronger feedback activity, more substantial discussion, or rework following submission. In such cases, the review appears to have contributed meaningfully to the integration outcome.

MRs labeled as *trivial* or *lightweight* are assigned the target label `can_self_approve = True`, indicating that they are historical candidates for reduced collaborative review or self-approval. MRs labeled as *substantive* are assigned `can_self_approve = False`, indicating that external review appears to have added observable value and should not be removed.

By construction, all MRs labeled as self-approvable have a pipeline failure rate of zero, since any MR with a failed pipeline is automatically assigned to the substantive class. This makes the target label deliberately conservative: the model is only asked to recommend reduced collaborative review for historical situations in which review added little observable value and delivery remained successful.

Creation-Time Features for Prediction

The prediction task must be performed at MR creation time, before any review activity has taken place. Accordingly, the model is trained exclusively on features that are available when the MR is opened. These features fall into three categories: the author's prior history, the structural characteristics of the submitted change, and the amount of context provided in the MR description. Table 9.6 summarizes the full feature set.

Importantly, no review-outcome variable such as comments, discussions, approvals, or post-submission commits is used as an input feature. Such variables are reserved exclusively for target annotation, ensuring that the prediction setting remains temporally valid.

Model Training and Evaluation

We train a single global prediction model on the combined labeled dataset from the ten Kaloom and TELUS projects to predict the binary target `can_self_approve` using the creation-time features listed in Table 9.6. Pooling the projects allows the model to learn patterns that are less dependent on a single review culture and more suitable for deployment across heterogeneous GitLab environments. Training one global model rather than one model per project is also a deliberate design choice for downstream tooling: it makes it possible to operationalize a single recommendation backend that can later be queried by a web extension exposing the prediction directly during MR inspection, without requiring per-project retraining or project-specific model selection at inference time.

Boolean variables are converted to binary integers and rows with missing values are excluded. For models that are sensitive to feature scale, standardization is applied within each bootstrap iteration to avoid information leakage. No explicit feature-selection step is applied at this stage.

We evaluate four classifiers: Decision Tree, Random Forest, Gradient Boosting, and Logistic Regression. Each model is trained and assessed over 100 bootstrap iterations. In each iteration, a bootstrap sample of the same size as the original dataset is drawn with replacement, the

Table 9.6 Features available at MR creation time used to train the self-approval recommendation model. These features are computable from the GitLab API for any project without requiring historical labeling on the target project

Category	Feature	Definition	Rationale
Author History	author_prior_failure_rate	Ratio of prior pipeline failures to prior pipeline executions.	Captures the author's historical delivery reliability.
	author_prior_pl_failures	Count of prior MRs with pipeline failures.	Provides an absolute measure of prior failure frequency.
	author_prior_mrs	Count of prior MRs in the project.	Contextualizes the author's history and experience.
	author_prior_merged	Count of prior merged MRs.	Reflects prior integration experience.
	author_prior_sa_rate	Proportion of prior MRs that were self-approved.	Captures prior self-approval behavior.
	author_prior_sm_count	Count of prior self-merged MRs.	Captures prior self-management behavior.
Change Characteristics	total_commits	Number of unique commits in the MR.	Provides a coarse indication of submission granularity.
	files_changed	Number of files modified.	Reflects the breadth of the submitted change.
	mr_additions	Total lines added.	Captures the size of newly introduced content.
	mr_deletions	Total lines deleted.	Captures the extent of removed content.
	mr_churn	Sum of additions and deletions.	Measures overall modification volume.
	new_files	Count of newly created files.	New files may require more conceptual review.
	deleted_files	Count of deleted files.	File removals may increase integration sensitivity.
	renamed_files	Count of renamed files.	Structural changes may increase review need.
	unique_directories	Distinct directories touched.	Measures the spread of the change across the project.
	unique_extensions	Distinct file extensions touched.	Reflects artifact diversity.
	touches_ci_config	Whether CI/CD configuration files are modified.	CI-related changes may increase delivery sensitivity.
	ci_config_files_touched	Count of CI configuration files modified.	Quantifies the extent of pipeline-related modifications.
Desc.	description_length	Character count of the MR description.	Proxies how much context is provided at submission.
	description_word_count	Word count of the MR description.	Complements description length as a context indicator.

model is trained on this sample, and performance is evaluated on the corresponding out-of-bag observations. For each iteration, we compute AUC, precision, recall, and F1. We then report the median and interquartile range across the 100 iterations.

This evaluation design provides a robust estimate of how well low-risk reduced-review candidates can be identified under repeated resampling, while preserving the distinction between training and testing data in each bootstrap round.

Results

Creation-time metrics appear sufficient to identify low-risk candidates for reduced collaborative review with reasonably strong predictive performance. As shown in Table 9.7, Gradient Boosting achieves the highest median AUC (0.829), followed closely by Random Forest (0.823). Decision Tree (0.779) and Logistic Regression (0.735) perform less well. The narrow interquartile ranges indicate that these results are stable across bootstrap samples.

The best-performing models show different precision–recall trade-offs. Gradient Boosting reaches the highest median precision (0.689), indicating that when it recommends a reduced-review path, the recommendation is correct in roughly seven cases out of ten according to the conservative historical label. However, its lower recall (0.648) means that it is more selective and leaves more potentially self-approvable MRs unrecommended. Random Forest provides a more balanced profile, with the highest recall (0.741) and the highest F1 score (0.683), suggesting that it identifies a larger share of low-risk candidates while maintaining acceptable recommendation quality.

By contrast, Logistic Regression performs noticeably worse than the tree-based models on all metrics. This suggests that the relationship between creation-time MR characteristics and reduced-review suitability is not purely linear, and that interactions between author history, change properties, and contextual features likely play an important role in the prediction task.

Taken together, these results indicate that low-risk candidates for reduced collaborative review can be predicted with useful accuracy from MR creation-time information alone. This is an important finding for the thesis because it shows that the transition from retrospective analysis to proactive recommendation is feasible. Rather than treating all self-approval or reduced-review behavior as equally risky, it becomes possible to support more selective and risk-aware review policies.

Table 9.7 Model comparison over 100 bootstrap iterations. Each cell reports the median value with the interquartile range in parentheses. The best value in each column is shown in boldface

Model	AUC	Precision	Recall	F1
Decision Tree	0.779 (0.005)	0.569 (0.014)	0.760 (0.031)	0.651 (0.009)
Random Forest	0.823 (0.004)	0.634 (0.010)	**0.741** (0.010)	**0.683** (0.006)
Gradient Boosting	**0.829** (0.004)	**0.689** (0.012)	0.648 (0.014)	0.669 (0.007)
Logistic Regression	0.735 (0.007)	0.512 (0.009)	0.780 (0.009)	0.619 (0.008)

9.3.3 Characteristics Driving the Reduced-Review Recommendation

RQ3: Which creation-time features have the strongest influence on the reduced-review recommendation

Approach

After establishing in RQ2 that low-risk reduced-review candidates can be identified with useful accuracy at creation time, we next examine which creation-time features drive these recommendations most strongly. The objective of this research question is not only to improve interpretability of the prediction model, but also to better understand what kinds of signals characterize MRs that are more likely to be suitable for reduced collaborative review or self-approval.

The analysis follows three steps. First, we extract feature-importance scores from the best-performing prediction model identified in RQ2, namely Gradient Boosting, across all 100 bootstrap iterations. This yields a distribution of importance values for each of the 20 creation-time features rather than a single model-specific ranking. Second, we convert these importance values into within-iteration feature ranks so that the stability of the relative ordering can be assessed across bootstrap samples. Third, we apply a Friedman test followed by Nemenyi post-hoc analysis to determine whether the observed ranking differences are statistically distinguishable.

More specifically, each bootstrap iteration produces one importance score per feature, after which features are ranked from most important to least important within that iteration. The

Friedman test is then used to assess whether the average feature ranks differ significantly across the 100 iterations. When this global test is significant, the Nemenyi post-hoc procedure is used to identify groups of features whose ranks are not significantly different at $\alpha = 0.05$. Features sharing the same group letter are therefore interpreted as belonging to the same importance tier. For interpretability, we report the mean rank and median importance for the ten highest-ranked features.

Results

Change size and author reliability emerge as the two dominant dimensions driving the recommendation. The amount of newly added content obtains the best average ranking across the 100 bootstrap iterations (mean rank 1.07) and the highest median importance (0.206), making it the strongest single predictor of whether an MR is recommended for a reduced-review path. The second-ranked feature is the author's prior pipeline failure rate (mean rank 2.63), which confirms the central finding observed in RQ1: the author's historical CI/CD reliability is a major signal of downstream risk. Taken together, these two features indicate that the recommendation model relies primarily on both the technical scope of the submitted change and the prior reliability of the contributor.

A second tier of influential features captures the structural complexity and spread of the MR. This group includes overall churn, the number of distinct directories touched, the length of the MR description, and the number of commits. Their presence near the top of the ranking suggests that the recommendation is sensitive not only to raw size, but also to how broadly the change is distributed in the codebase and how much contextual information is provided at submission time. In particular, the strong position of MR description length suggests that richer descriptions may be associated with situations in which the submitted change is easier to understand and therefore more likely to be judged as suitable for a lighter review path.

Author-related behavioral history also remains an important part of the recommendation signal. The model gives notable weight to the author's prior self-merging count, prior self-approval

rate, prior pipeline failures, and prior MR experience. This indicates that the model does not rely solely on immediate change characteristics. It also considers whether the contributor has a history of reliable integration, previous self-managed behavior, and broader project experience. In this sense, the recommendation is influenced not only by what the MR looks like at submission time, but also by who is submitting it.

By contrast, several categorical or process-specific indicators contribute less strongly to the prediction. Modifying CI/CD configuration files, renaming files, and other lower-ranked structural flags do not appear among the dominant predictors. This suggests that the model relies more on continuous risk-related signals, such as size, spread, and prior reliability, than on isolated categorical markers. In practical terms, the recommendation appears to be shaped less by any single special-case condition and more by the overall risk profile of the MR and its author.

Taken together, these results show that the reduced-review recommendation is driven primarily by two broad dimensions: characteristics of the submitted change and characteristics of the contributor's prior track record. This is an important result for the chapter because it clarifies the basis on which proactive recommendation becomes possible. Rather than treating self-approval as a purely procedural decision, the model operationalizes reduced collaborative review as a risk-sensitive judgment grounded in both MR complexity and author reliability.

9.4 Discussion

Taken together, the results of this chapter suggest that downstream delivery reliability cannot be understood through self-managed peer review behavior alone. Across the studied projects, the most stable signal associated with pipeline failure is not whether a merge request was self-approved or self-merged, but the author's prior delivery reliability. In RQ1, history-based indicators emerge as the strongest and most consistent factors associated with downstream CI/CD failure, whereas self-review decision variables show weaker and less stable relationships across projects. This shifts the interpretation of review risk away from a simple distinction between self-managed and externally reviewed merge requests. Instead, delivery risk appears to depend

more fundamentally on who is making the change, under what prior reliability conditions, and within which project context.

This perspective is important because self-managed review is often treated as a problematic or inherently unsafe practice. Such caution is understandable: self-approval and self-merging reduce independent scrutiny and may create concern about weaker validation. At the same time, self-management can also save time, reduce coordination overhead, and help teams avoid unnecessary review delays, especially when changes are simple, routine, or made by contributors with a strong history of reliable delivery. Our results suggest that these two views need to be reconciled. Self-management should not be treated as universally bad practice, but neither should it be adopted without care. Rather, it should be understood as a process choice whose safety depends on context. When teams rely on self-managed review, they need to remain aware of the associated risk conditions, including contributor history, change characteristics, and the presence or absence of strong automation safeguards.

This helps explain why self-managed review behavior does not exhibit a uniform relationship with failure. Self-approval and other self-managed conditions are not consistently associated with poorer outcomes across projects. Even when self-merging is statistically significant, its direction differs depending on the project. In some contexts, self-managed behavior is associated with higher failure risk, whereas in others it is associated with lower risk or shows no clear difference. This heterogeneity indicates that self-management is not, by itself, a reliable proxy for unsafe delivery. Rather, its meaning depends on the surrounding process environment, including local review norms, automation safeguards, and contributor history. From the perspective of peer review effort optimization, this is an important result: reducing collaborative review effort does not automatically imply greater downstream risk, and blanket assumptions about self-managed review are therefore difficult to support empirically.

The results also suggest that review effort should be interpreted together with the role of automation. While collaborative review intensity shows only moderate and project-dependent associations with failure, pipeline-gated integration appears more consistently protective in

several contexts. In particular, enabling merge when the pipeline succeeds is associated with lower failure risk in multiple projects, especially in the TELUS setting. This indicates that part of the safety often attributed to peer review may in practice be carried by automation safeguards rather than by uniformly strong human review involvement. Peer review remains important, but its role should be understood as part of a larger socio-technical process in which automation and human judgment jointly shape delivery outcomes.

The findings of RQ2 extend this discussion by showing that candidates for reduced collaborative review can be identified with useful accuracy at creation time. Using a conservative annotation strategy, the chapter operationalizes such candidates as merge requests whose historical reviews produced little or no observable contribution and whose pipelines succeeded. On that basis, the predictive results show that creation-time information is sufficient to distinguish many such cases from merge requests that appear to require more substantive review. The best-performing models achieve AUC values above 0.82, demonstrating that the optimization perspective of the thesis can move beyond retrospective explanation toward proactive support. At the same time, these results should not be interpreted as evidence that peer review can be replaced wholesale. The labeling strategy captures absence of observable review contribution in recorded traces, not proof that review had no value whatsoever. Silent validation, informal discussion, and off-platform coordination may still play an important role. The appropriate interpretation is therefore one of decision support: the goal is to allocate review attention more selectively, not to eliminate human judgment.

The feature-importance analysis of RQ3 further clarifies why such prediction is possible. The recommendation model is driven primarily by two broad dimensions: the characteristics of the submitted change and the contributor's prior track record. On the one hand, the model is sensitive to the size, spread, and structural complexity of the merge request. On the other hand, it relies heavily on signals of historical reliability, prior pipeline behavior, and previous self-management patterns. This is consistent with the results of RQ1 and reinforces the chapter's overall interpretation: the question of whether reduced collaborative review is acceptable is not primarily procedural, but risk-sensitive. The model does not rely mainly on isolated categorical

flags; rather, it combines multiple signals that together describe whether a merge request appears suitable for lighter review treatment.

The prototype web extension discussed in this chapter translates these empirical results into a more practical workflow. By exposing the recommendation at the point where developers and reviewers inspect the merge request, it brings the analysis closer to the actual review decision. This is important because review-effort adaptation is only useful if it can inform the moment at which teams decide how much scrutiny a change requires. In this sense, the extension should be interpreted as a decision-support layer rather than as an automated approval mechanism. It can highlight merge requests that appear suitable for reduced collaborative review, while leaving the final decision to the team and preserving sensitivity to local policies, criticality, and domain knowledge.

More broadly, this chapter contributes to the dissertation by showing that downstream delivery reliability provides an essential perspective for reasoning about peer review effort. The findings challenge both the assumption that stronger collaborative review is always the safest option and the opposite assumption that self-approval is inherently risky. Instead, the evidence supports a more nuanced view: external review remains important, but its necessity varies according to the risk profile of the merge request, the history of its author, and the surrounding process safeguards. In this way, the chapter extends the dissertation's process-oriented view of peer review effort from upstream planning conditions to downstream delivery outcomes.

A natural next step is to enrich this predictive perspective with models that can reason over context beyond numerical metrics alone. In future work, we plan to integrate large language models into the recommendation pipeline in order to capture richer semantic and contextual signals from merge request titles, descriptions, commit messages, review comments, file paths, and other textual or structural cues. Such an extension would complement the current metric-based models by allowing the prediction process to consider not only the measurable properties of a merge request, but also the meaning and intent of the change. This could improve prediction

Table 9.8 Effort savings potential per project. *Wasted reviews* are peer-reviewed MRs annotated as self-approvable (trivial or lightweight review). Time to complete peer review is measured as hours. Working days assume 8-hour days. Pipeline failure rate among wasted reviews is 0% in all projects by construction

Project	Review Count			Effort Saved		
	Peer Rev.	Wasted	% Wasted	Med. (h)	Total (h)	Work Days
CP	7,636	3,302	43.2	0.9	29,734	3,717
DP	6,272	1,298	20.7	1.0	9,560	1,195
MP	5,941	2,202	37.1	0.9	17,594	2,199
FPGA	539	161	29.9	0.6	3,531	441
PF	3,524	1,638	46.5	0.9	16,640	2,080
NetApps	1,511	532	35.2	0.0	1,658	207
Platform	931	347	37.3	0.0	2,260	282
Infrastructure	93	16	17.2	0.0	0	0
CDAF	378	128	33.9	0.0	301	38
CNE-IP	131	94	71.8	—	146	18

quality, provide more interpretable recommendations, and better support risk-aware peer review adaptation in practice.

9.5 Implications

9.5.1 Implications for our industrial partners

An important practical consequence of these results is that part of the current peer review activity may be avoidable, at least in the sense that it shows little or no observable contribution in the recorded MR traces. Table 9.8 shows that, across the studied projects, a substantial proportion of peer-reviewed MRs were later annotated as candidates for reduced collaborative review, meaning that their recorded reviews were trivial or lightweight and were not accompanied by observed pipeline failure. This proportion ranges from 17.2% to 71.8% depending on the project. In Kaloom, the percentage varies from 20.7% in DP to 46.5% in PF, whereas in TELUS it ranges from 17.2% in Infrastructure to 71.8% in CNE-IP. These numbers do not prove that external review had no value in a strict causal sense, but they do suggest that the current allocation of review attention may be broader than necessary in some contexts.

The magnitude of this potential saving is particularly visible in the Kaloom projects, where the cumulative review-related elapsed time associated with these low-risk candidates exceeds 77,000 hours. Although time to complete peer review should not be interpreted as a direct measure of true reviewer effort, since it also captures waiting time and coordination delays, it still provides a useful estimate of the review-related time that may be reduced if low-risk MRs are redirected toward lighter approval paths. In the TELUS projects, the median time to complete peer review is often near zero, so the estimated time savings are smaller in absolute terms even when the proportion of reduced-review candidates remains high. This pattern suggests that the practical benefit of the recommendation may vary across organizations: in some settings, it may primarily reduce elapsed time to complete peer review, whereas in others it may mainly reduce unnecessary review overhead and free reviewer attention for more complex changes.

The relationship between review routinization and project stability is also noteworthy. Projects with lower observed pipeline-failure rates sometimes show higher proportions of reviewed MRs that are later annotated as reduced-review candidates. For example, project CNE-IP combines the lowest failure rate with the highest proportion of peer-reviewed MRs that fall into this category. This pattern suggests that, in relatively stable environments, a larger share of peer review may function as confirmation rather than correction. Under such conditions, a risk-aware recommendation system may help teams reserve external review for cases where it is more likely to add value, while allowing straightforward and historically low-risk MRs to proceed with lighter review mechanisms.

For industrial partners, the prototype web extension offers a practical path toward operationalizing this idea. By providing a live recommendation during MR inspection, the extension can make risk-aware review allocation visible at the moment when developers and reviewers decide whether additional scrutiny is needed. This does not require organizations to remove review gates globally. Instead, it supports a more controlled process in which teams can define local thresholds, inspect the model's recommendation, and decide whether an MR should receive standard collaborative review, lighter review, or explicit self-approval. Such a tool can therefore support policy experimentation without replacing team judgment.

More broadly, these results support a move from uniform review governance toward context-sensitive review allocation. The evidence does not suggest that external review should be removed, nor that self-approval is universally safe. Rather, it suggests that peer review effort can be distributed more selectively if delivery risk is assessed using the combined signals identified in this chapter. In practical terms, this means that low-risk MRs submitted by historically reliable contributors and characterized by limited scope may not always need the same degree of external review as larger, more dispersed, or historically riskier changes. Such a shift would align peer review policy more closely with the actual distribution of delivery risk.

9.5.2 Implications for researchers

For researchers, this chapter reframes self-approval as a context-dependent manifestation of reduced collaborative review rather than as a binary risk factor. This has several methodological consequences for future empirical work on peer review and delivery reliability. The observation that a substantial share of peer-reviewed MRs are later annotated as low-risk reduced-review candidates, yet do not exhibit observable pipeline failure, suggests that studies treating self-approval as uniformly risky may produce biased conclusions. Future empirical studies on peer review should therefore stratify their analyses by delivery-risk profile rather than evaluating self-approval in isolation. They should also explicitly report the operational rules used to define self-management, self-approval, and self-merging, since these conventions vary across organizations and directly affect the transferability of results.

The chapter also contributes a measurement perspective that researchers can extend. Linking MR-level peer review decisions to CI/CD outcomes provides an outcome anchor that complements internal review process metrics such as comment counts, review duration, or reviewer engagement. This supports a broader research practice in which peer review studies are validated against downstream delivery signals, which are less directly confounded by reviewer availability or organizational urgency than process-internal metrics. The creation-time recommendation framework, which reaches an AUC above 0.82, further illustrates that MR-level decisions can be studied as prediction problems anchored in downstream outcomes. This framing can be

generalized to other review-related decisions, such as reviewer assignment, fast-track approval, escalation triggers, or review-depth recommendation.

Finally, the observation that the practical value of reduced-review recommendation varies across organizational contexts motivates multi-site replication studies that explicitly report context characteristics alongside the main effect estimates. Such characteristics include baseline pipeline-failure rates, the level of review routinization, approval conventions, self-management norms, and the strength of CI/CD gating mechanisms. Reporting these contextual factors would make it easier to compare and aggregate findings across organizations and to understand when reduced collaborative review is likely to be acceptable.

9.6 Threats to Validity

This chapter is subject to several threats to validity. First, the notion of a self-managed or reduced-review MR depends on the operational rules used to define self-management, self-approval, self-merging, and limited external review from GitLab traces. Different organizations may use different review and approval conventions, which may affect the transferability of the labeling rules.

Second, the current analysis of downstream reliability focuses primarily on pipeline failure. Although this is an important indicator of delivery reliability, it does not capture all possible downstream consequences, such as latent defects, rollback frequency, post-release incidents, maintainability degradation, or security issues. Therefore, the term “low risk” should be interpreted as low observed CI/CD delivery risk under the available traces, not as a complete guarantee of software quality.

Third, the annotation strategy used for the recommendation model is conservative but trace-based. MRs labeled as candidates for reduced collaborative review are those for which the recorded review produced little observable feedback and the pipeline succeeded. However, reviewers may still provide silent validation, communicate outside the platform, or detect risks that are not

visible in the MR traces. Consequently, the label should be interpreted as an approximation of observable review value, not as proof that external review was unnecessary.

Fourth, the analyses are observational and do not control for all potential confounding factors, such as contributor expertise, subsystem criticality, change urgency, informal communication, team ownership, or release pressure. Consequently, observed associations should not be interpreted as direct causal effects. The recommendation model should therefore be understood as decision support for review allocation rather than as a causal model of review effectiveness.

Finally, the study is conducted in industrial GitLab contexts from Kaloom and TELUS. This strengthens the industrial relevance of the findings, but may limit generalizability to other review platforms, project structures, approval policies, or CI/CD configurations. Replications in additional organizations and open-source settings would help determine how stable the observed patterns are across contexts.

9.7 Conclusion

This chapter examined the relationship between peer review decisions and downstream CI/CD reliability. Its main objective was to determine whether reduced collaborative review, observable through self-managed decisions such as self-approval, can be treated as a risk-aware adaptation of peer review effort rather than as a uniformly unsafe practice.

The results show that self-managed review decisions are not consistently associated with poorer pipeline outcomes across the studied projects. Instead, downstream failure is more strongly and more consistently associated with the author's prior delivery reliability, together with project-specific safeguards and change characteristics. This finding challenges the assumption that all MRs require the same level of external review and supports a more selective view of peer review effort allocation.

The chapter also shows that low-risk candidates for reduced collaborative review can be identified at MR creation time with useful predictive performance. Using a conservative historical label

based on trivial or lightweight review outcomes and successful pipeline execution, the best-performing models achieve AUC values above 0.82. The recommendation is driven mainly by change size, structural spread, submission context, and the contributor's prior track record. These results indicate that the suitability of reduced collaborative review is not a simple procedural property, but a risk-sensitive judgment that combines MR characteristics with author history.

Finally, the chapter introduces a prototype web extension that provides live recommendations during MR inspection. This prototype illustrates how the proposed model can be integrated into developers' existing workflow to support risk-aware review-effort adaptation in practice. From the perspective of the thesis, these findings complete the process-oriented view of peer review effort by linking review decisions to downstream delivery reliability. They show that the challenge is not simply to avoid self-managed review behavior, but to determine under which conditions collaborative review effort can be reduced while preserving the safeguards needed for reliable DevOps delivery.

CHAPTER 10

REVMINE: AN LLM-ASSISTED TOOL FOR CODE REVIEW MINING AND ANALYSIS ACROSS GIT PLATFORMS

Samah Kansab[1] , Oussama Cherguelaine[2] , Merabet Mohammed Riad[2] , Francis Bordeleau[1] , Ali Tizghadam[3]

[1] Department of Software Engineering and IT, École de Technologie Supérieure, 1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[2] École Nationale Supérieure D'Informatique, Oued Smar, Alger, Algérie
[3] TELUS, 25 York St, Toronto, Ontario, Canada M5J 2V5

Paper Published in the 35th IEEE International Conference on Collaborative Advances in Software and Computing on November 2025; extended to the Empirical Software Engineering Journal on May 2026

Preamble

This chapter presents RevMine, an LLM-assisted tool that streamlines DevOps-pipeline mining across GitHub and GitLab. RevMine accepts natural-language data-collection requests, translates them into validated extraction plans, executes them against the appropriate platform APIs, and computes a literature-grounded set of metrics. Although it originated as a peer review mining tool, its scope in this thesis version covers the three DevOps pipeline stages studied in the preceding chapters: Kanban-style planning, peer review, and CI/CD delivery. We evaluate RevMine along four dimensions: technical correctness of data collection, accuracy of LLM orchestration, dependency-completeness of generated plans, and perceived usefulness through an external evaluation. Together, these analyses establish RevMine as a reproducible infrastructure for operationalizing the process analyses developed throughout this thesis.

Compared with the CASCON version, the chapter version adds a fully implemented tool, with empirical study. Compared to the submitted journal, this chapter adapt introduction, discussion and conclusion to fit thesis context.

10.1 Introduction

The preceding chapters showed that peer review effort in DevOps workflows cannot be understood as an isolated activity. It is observable through MR traces, varies across MR types and workflow conditions, is influenced by upstream planning, and is connected to downstream CI/CD outcomes.

Realizing this analytical agenda, however, requires collecting and combining heterogeneous data from multiple parts of the DevOps workflow, including MR metadata, commits, file changes, comments, approvals, issue histories, workflow transitions, and pipeline outcomes. Although such data is available through platforms such as GitHub and GitLab, its practical use remains difficult. Researchers and practitioners must handle authentication, pagination, rate limits, platform-specific APIs, and the transformation of raw data into analysis-ready datasets. As a result, repository mining often depends on ad hoc scripts, which can reduce reproducibility and make analyses difficult to transfer across projects, platforms, or organizations.

This tooling challenge is central to the objective of this thesis. If peer review effort is a multi-dimensional process phenomenon, then a supporting tool must do more than collect isolated pull request or MR records. It must help users reconstruct the broader analytical chain linking upstream planning, peer review activity, and downstream delivery. This includes, for example, combining MR creation-time features with commits and discussions, classifying artifact types, detecting workflow deviations, and connecting review decisions to CI/CD pipeline outcomes.

This chapter presents *RevMine*, an LLM-assisted tool for reproducible DevOps-pipeline data collection and analysis across GitHub and GitLab. RevMine allows users to express collection and analysis needs in natural language or through manual configuration, translates these needs into structured collection plans, validates the required fields and dependencies, executes the corresponding API calls, computes curated process metrics, and exports analysis-ready datasets. Although RevMine originated as a peer review mining tool, the thesis version extends its scope to the three DevOps stages studied throughout this dissertation: planning, peer review, and CI/CD

delivery. In this sense, RevMine operationalizes the upstream–review–downstream perspective developed in the preceding chapters.

The objective of this chapter is to describe and evaluate RevMine as a reproducible tool for DevOps-pipeline mining and peer review analytics. More specifically, the chapter examines whether RevMine can collect pull request and MR data correctly across GitHub and GitLab, whether its LLM orchestration layer can accurately translate natural-language requests into valid collection plans, whether these plans include the metric dependencies required for downstream analysis, and how potential users perceive the tool's usefulness and adoption value.

The main contributions of this chapter are:

- RevMine, a fully implemented, open-source tool that supports reproducible DevOps-pipeline data collection and analysis across GitHub and GitLab;
- an LLM-assisted orchestration mechanism that translates natural-language data-collection requests into structured and validated extraction plans, while preserving a manual configuration mode for reproducibility-sensitive workflows;
- a literature-grounded catalogue of peer review, change, planning, and CI/CD metrics that maps high-level empirical software engineering concepts to concrete platform data and derived features;
- a systematic evaluation of RevMine's data collection correctness across GitHub and GitLab repositories, comparing its outputs with hand-written mining scripts;
- a systematic evaluation of the LLM orchestration layer using a benchmark of natural-language scenarios, multiple model configurations, and complementary exact-match, subset-match, and overlap-based measures;
- an evaluation of feature-to-metric dependency completeness, assessing whether generated collection plans include the raw data required to compute the requested metrics;
- an external formative evaluation with research and industry participants, assessing perceived usefulness, trust, and adoption intent;
- an extension of RevMine from peer review mining to cross-stage DevOps analytics, enabling joins between Kanban-style planning data, PR/MR review data, and CI/CD delivery outcomes.

10.2 RevMine: Tool Description

This section presents RevMine's design rationale, system architecture, collected data, and supported analyses. RevMine is a tool for collecting, structuring, and analyzing peer review and DevOps process data from GitHub and GitLab through a unified workflow that combines platform-aware data extraction with LLM-assisted request interpretation. Beyond raw extraction, RevMine exposes an explicit catalogue of collected fields and derived metrics, together with a set of supported analyses that operationalize the constructs developed in the preceding chapters of this thesis.

10.2.1 Design Rationale

RevMine was designed to address three limitations commonly encountered in peer review mining workflows.

Accessibility. Existing data collection pipelines require researchers to understand platform-specific APIs, authentication mechanisms, pagination behavior, and data transformation steps. RevMine reduces this burden by allowing users to express collection or analysis needs through either natural language or a manual configuration interface.

Platform abstraction. GitHub and GitLab expose review data through different endpoints, schemas, and terminology. RevMine hides these differences behind a unified internal representation, allowing the same analytical workflow to be reused across platforms with minimal user-side adaptation.

Reproducibility. RevMine separates raw data acquisition from downstream transformation and analysis. Retrieved platform responses are preserved before further processing, enabling re-execution of filtering, feature computation, and analysis without repeating API calls. This design supports reproducible empirical studies while reducing dependence on platform rate limits and endpoint volatility.

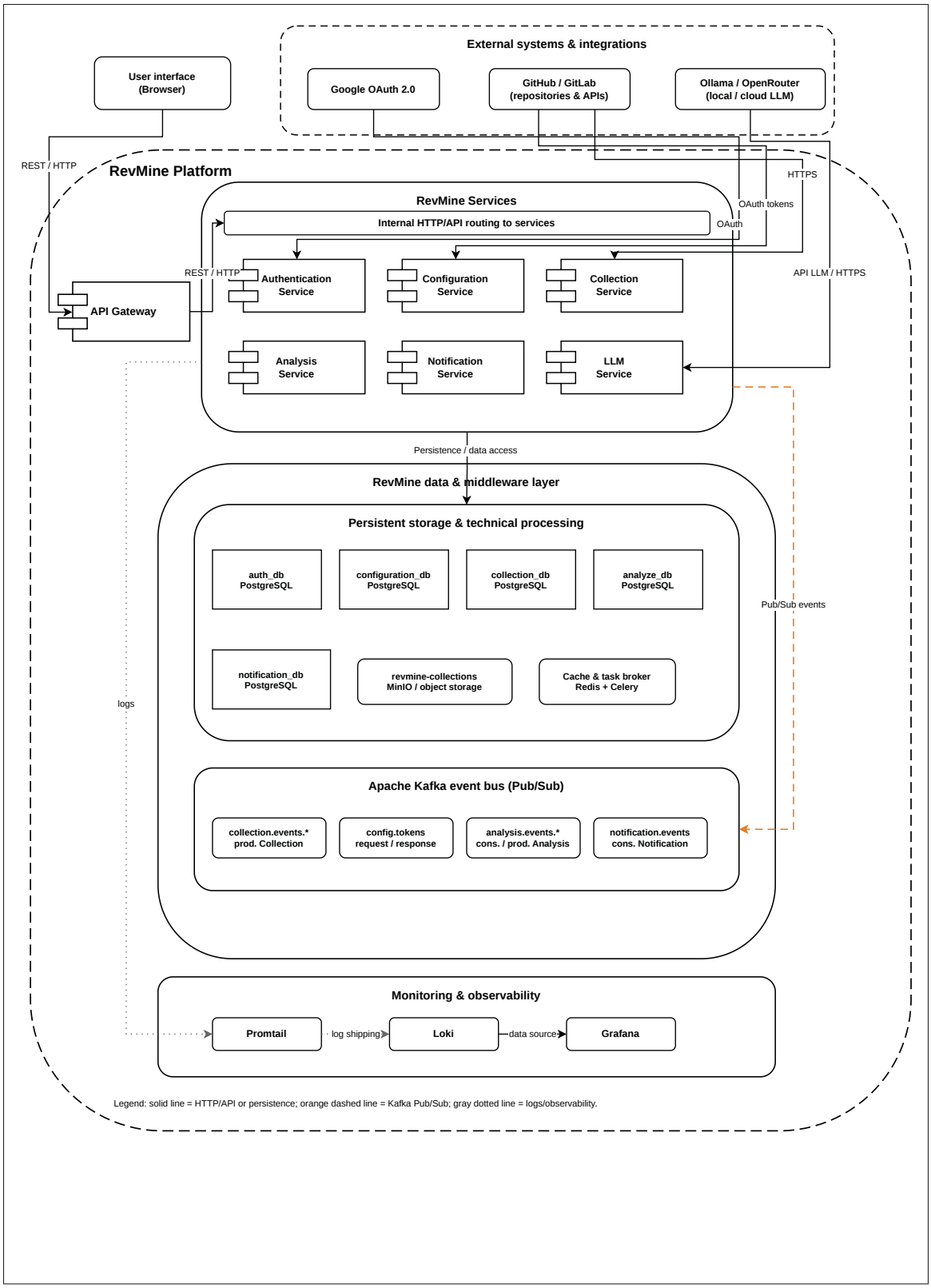


Figure 10.1 Overview of RevMine's microservice-based layered architecture

10.2.2 System Architecture

RevMine is implemented as a **microservice-based layered architecture**. This architecture separates user interaction, request routing, authentication, configuration, data collection, LLM-based interpretation, analysis, notification, persistence, event distribution, and monitoring into distinct components. Rather than centralizing these responsibilities in a monolithic application, RevMine organizes them around independent services connected through REST/HTTP calls, persistence interfaces, asynchronous tasks, and publish/subscribe events. Figure 10.1 summarizes the main architectural components and their interactions.

User interface and external integrations

The user-facing layer consists of a browser-based interface through which users interact with RevMine. The browser communicates with the platform through REST/HTTP requests. RevMine also interacts with several external systems: GitHub and GitLab provide repository and API data; Google OAuth 2.0 supports external authentication; and Ollama or OpenRouter provide local or cloud-based LLM access. These external integrations remain outside the RevMine platform boundary, while the platform encapsulates the services and storage components needed to process their data.

API gateway and routing layer

All requests from the user interface enter RevMine through the *API Gateway*. The gateway provides a single entry point to the platform and routes REST/HTTP requests to the appropriate internal services. For protected routes, the gateway delegates identity validation to the *Authentication Service* and forwards the authenticated user context to downstream services. This keeps the gateway focused on routing and access control while isolating authentication logic in a dedicated service.

Service layer

The core functionality of RevMine is implemented through six main microservices. The *Authentication Service* manages login, user identity, token issuance, token refresh, and OAuth callback handling. The *Configuration Service* manages platform configuration and access tokens required for repository integrations. The *Collection Service* extracts data from external platforms such as GitHub and GitLab. The *Analysis Service* computes metrics and produces analytical outputs from collected data. The *Notification Service* manages user and system notifications. The *LLM Service* interprets natural-language requests and communicates with local or cloud LLM providers. These services are reachable through internal HTTP/API routing and can also exchange events through the Kafka event bus.

Data and middleware layer

RevMine uses a service-oriented data management strategy. Each main service has a dedicated PostgreSQL database, which limits coupling between services and supports independent evolution of the platform. The architecture includes `auth_db`, `configuration_db`, `collection_db`, `analyze_db`, and `notification_db`. In addition, raw and semi-structured collected artifacts are stored in `revmine-collections`, a MinIO object store. This separation distinguishes structured operational data from larger JSON-based repository artifacts.

Task execution and caching

RevMine integrates Redis and Celery as a cache and task broker layer. Redis acts as the broker and result backend, while Celery workers execute asynchronous jobs outside the synchronous request-response path. This design is useful for long-running operations such as large-scale collection, analysis generation, or batch processing, because the user interface can remain responsive while background tasks continue to execute.

Event distribution layer

RevMine uses Apache Kafka as a publish/subscribe event bus. Kafka decouples services that produce events from services that consume them, which reduces synchronous dependencies between components. The architecture defines several event channels, including `collection.events.*`, `config.tokens.request/response`, `analysis.events.*`, and `notification.events`. For example, the Collection Service can produce collection events, the Analysis Service can consume and produce analysis events, and the Notification Service can consume notification events. This event-oriented layer supports asynchronous coordination between services.

Monitoring and observability layer

RevMine includes a monitoring and observability stack composed of Promtail, Loki, and Grafana. Promtail collects service logs and ships them to Loki. Loki stores and indexes the logs, while Grafana provides the visualization and querying interface. This layer gives operators visibility into platform execution and supports debugging, monitoring, and maintenance across the different services.

Architectural summary

Overall, the new architecture separates RevMine into a browser-based user interface, external integrations, a gateway-based access layer, specialized microservices, independent persistence backends, a Redis/Celery task layer, a Kafka-based event bus, and a centralized observability stack. This separation is important for RevMine because natural-language interaction, repository extraction, authentication, metric computation, asynchronous processing, and monitoring have different execution patterns, scalability needs, and failure modes.

10.2.3 Collected Data and Supported Metrics

RevMine collects and derives metrics from PR/MR traces to support empirical analysis of code review activity, collaboration, change complexity, project evolution, upstream workflow

behavior, and downstream CI/CD delivery. The metric catalogue is organized by analytical purpose rather than by API endpoint. This organization helps users move from a research question to the corresponding data requirements, while keeping the definition and motivation of each metric explicit.

Table 10.1 summarizes the main metric categories currently supported by RevMine. These metrics cover PR/MR metadata, commits, code churn, modified files, comments and discussions, formal reviews, collaboration, code quality, time-based indicators, Kanban workflow metrics, and CI/CD delivery metrics.

This catalogue makes the operationalization of RevMine metrics explicit. Each metric is linked to a concrete aspect of PR/MR activity, workflow execution, or delivery behavior. For example, lead time and cycle time capture the temporal duration of the review process, while pickup time and time to first review capture review responsiveness. Similarly, comments, discussions, formal reviews, inline review comments, reviewers, participants, and committers describe complementary dimensions of collaboration. Code churn, initial size, rework size, modified files, file types, and historical entropy provide change-based indicators of effort and complexity. Finally, Kanban and CI/CD metrics extend the analysis beyond the PR/MR itself by connecting code review with upstream planning and downstream delivery.

10.2.4 Supported Analyses

Beyond metric extraction, RevMine supports a set of predefined and configurable analyses designed to help users explore code review practices, development activity, collaboration patterns, change complexity, workflow behavior, and delivery performance. These analyses are implemented on top of the metric catalogue described above and can be generated individually, through custom axes, or in bulk.

Table 10.2 summarizes the main analyses currently supported by RevMine, together with their purpose.

Table 10.1 Main metrics supported by RevMine and their analytical purpose

Category	Metrics	Description	Analytical purpose
PR/MR metadata	PR/MR identifier, title, author, state, creation date, merge date, close date, merged by	Basic descriptive fields that identify each PR/MR, its creator, its lifecycle state, and the user who performed the merge.	Provides the core unit of analysis and enables filtering, grouping, lifecycle analysis, and outcome-based comparisons.
Commit metrics	Number of commits, committers, number of unique committers, mean time between commits, commit SHA, commit message, commit author, commit date, commit changes	Captures both aggregate and per-commit information associated with a PR/MR.	Supports analysis of change granularity, contributor activity, development rhythm, and commit-level evolution.
Code churn metrics	Additions, deletions, churn additions, churn deletions, initial PR/MR size, rework size	Measures the number of added and deleted lines, both globally and across commits, and distinguishes changes prepared before review from changes made after review feedback.	Approximates development effort, review-induced rework, and the size of the submitted and revised changes.
File metrics	Number of modified files, file types, file name, file status, file additions, file deletions, file changes	Captures the files affected by a PR/MR, their extensions, their statuses, and their per-file change size.	Supports scope analysis, artifact-sensitive analysis, and identification of the technical areas most affected by changes.
Comment and discussion metrics	Number of comments, number of discussions, PR comments, comment author, comment date, comment body, discussion notes, resolved discussions	Captures general PR/MR comments, discussion threads, comment metadata, and whether discussions were resolved.	Measures review interaction, discussion intensity, and collaborative exchange during code review.
Review metrics	Number of formal reviews, review comments count, review state, review author, review date, review body, inline comment body, inline comment author, inline comment date, inline comment position, inline comment path	Captures structured review events and inline code review comments, including approval, change-request, and comment-only review states.	Supports analysis of formal review decisions, code-level feedback, approval behavior, and review depth.
Reviewer and collaboration metrics	Number of reviewers, reviewer list, number of participants, number of committers, number of discussion participants, minor authors, major authors	Measures the different actors involved in a PR/MR, including reviewers, committers, discussion participants, and contributors with minor or major commit activity.	Characterizes collaboration intensity, work distribution, participation structure, and concentration of contribution.
Code quality and complexity metrics	Historical entropy	Measures the dispersion of file changes across the codebase.	Supports the assessment of change scattering and structural complexity.
Time-based metrics	Lead time, delta time, pickup time, time to first review, first review timestamp, first comment timestamp, approval timestamp, review duration, approval time, cycle time	Captures the temporal evolution of a PR/MR from creation to first interaction, review, approval, closure, or merge.	Supports analysis of review responsiveness, review duration, approval delay, lifecycle speed, and process bottlenecks.
Kanban workflow metrics	Kanban lead time, Kanban cycle time, throughput, work in progress, cumulative flow diagram, time per column, blocked ratio, assignee load	Captures upstream workflow behavior from Kanban-style project management data.	Enables analysis of planning flow, workload, blocked work, throughput, and upstream conditions that may influence review activity.
CI/CD metrics	CI/CD success rate, build duration, failure rate by job, mean time to recovery, deployment frequency, queue time, runner utilization, flaky jobs	Captures downstream delivery and pipeline execution behavior.	Supports analysis of delivery reliability, build performance, operational delay, and the relationship between review decisions and CI/CD outcomes.

RevMine also supports several analysis modes and visualization options. Users can generate predefined metrics by specifying a metric code, define custom X/Y axes, or launch multiple analyses in bulk. The platform supports common aggregation functions, including sum, mean, median, count, minimum, maximum, and standard deviation. Time-based analyses can be grouped by day, week, month, quarter, or year. The supported chart types include line, bar,

Table 10.2 Supported analyses in RevMine and their analytical purpose

Analysis	Description	Analytical purpose
Commits over time	Aggregates commits by time period and visualizes them using line, bar, or area charts.	Identifies periods of high or low development activity.
PR/MR creation timeline	Displays the number of PRs/MRs created per time period.	Provides a global view of project activity and review workload over time.
Lead time distribution	Shows the distribution of review completion times using histograms or box plots.	Helps detect delayed reviews, bottlenecks, and inefficient review patterns.
Commit distribution	Shows the distribution of the number of commits per PR/MR.	Supports analysis of work granularity and submission practices.
PR/MR size analysis	Analyzes the initial size of PRs/MRs in terms of changed lines.	Helps identify large submissions and compare the size of reviewed changes.
Files modified analysis	Examines the number of files modified per PR/MR.	Provides insight into the scope of changes and the breadth of affected artifacts.
Committer analysis	Analyzes the number of unique contributors per PR/MR and the activity of committers.	Supports the study of work distribution and contributor involvement.
Code churn analysis	Compares additions and deletions using dual histograms or related visualizations.	Estimates development effort and identifies large or unstable changes.
Discussion analysis	Studies the number of discussions and comments associated with PRs/MRs.	Supports analysis of review interaction, discussion complexity, and collaborative engagement.
Review-time analyses	Includes pickup time, time to first review, review duration, approval time, and cycle time analyses.	Identifies review responsiveness, approval delays, and process bottlenecks across the PR/MR lifecycle.
Collaboration analysis	Produces multi-panel analyses of participants, reviewers, committers, and discussion participants.	Characterizes collaboration intensity and the distribution of roles in the review process.
Top actors analysis	Ranks top reviewers, authors, and committers according to their activity.	Identifies key actors in the project and supports participation analysis.
File-type distribution	Shows the most frequent modified file extensions.	Identifies dominant technologies and supports artifact-sensitive analysis.
Entropy analysis	Measures the historical entropy of file changes.	Supports the assessment of change complexity and scattering.
Rework analysis	Analyzes rework size and the proportion of changes performed after review feedback.	Captures corrective effort and supports analysis of review-induced modifications.
PR/MR complexity analysis	Computes a composite complexity score based on commits, files, discussions, and participants.	Provides a synthetic view of PR/MR complexity and helps distinguish simple from complex changes.
Correlation matrix	Computes and visualizes correlations between numeric metrics using heatmaps.	Helps identify associations between variables, such as size, churn, collaboration, and review duration.
Churn scatter analysis	Visualizes additions against deletions.	Helps identify change patterns, outliers, and unusually large or asymmetric changes.
State distribution	Shows the distribution of PR/MR states, such as open, closed, or merged.	Supports outcome analysis and project-level monitoring of contribution status.
Project comparison	Compares mean, median, or sum values across projects.	Enables cross-project analysis of review activity, complexity, collaboration, and delivery behavior.
Kanban analyses	Includes Kanban lead time, cycle time, throughput, work in progress, cumulative flow diagrams, column time, blocked ratio, and assignee load.	Supports analysis of upstream workflow efficiency, planning flow, workload distribution, and blocked work.
CI/CD analyses	Includes success rate, build duration, failure rate by job, mean time to recovery, deployment frequency, queue time, runner utilization, and flaky job analysis.	Supports downstream delivery analysis and links code review activity with pipeline reliability and delivery performance.
Custom chart analysis	Allows users to define their own X and Y axes from available fields.	Supports exploratory analysis beyond predefined metrics.
Bulk analysis	Allows users to generate multiple analyses in one request.	Facilitates reproducible and efficient generation of complete analysis suites.

scatter, area, pie, histogram, box plot, and heatmap. Users can also filter analyses using keywords on text fields, date ranges, and custom column-value filters.

Taken together, the supported metrics and analyses position RevMine as an analysis-oriented platform for PR/MR mining. The tool does not only extract raw repository data; it structures

this data into interpretable measures, predefined visual analyses, and configurable analytical views. This design supports both exploratory use, where users inspect project activity and review behavior through dashboards, and empirical research use, where the same metrics can be exported and reused in reproducible studies.

10.3 Study Design

This section presents the empirical study design used to evaluate RevMine as a fully implemented tool. We structure the evaluation around three research questions, each targeting a distinct dimension of the tool's correctness, intelligence, and breadth. The external evaluation reported in Section 10.5 complements these questions with formative feedback from researchers and practitioners outside the development team.

10.3.1 Research Questions

The chapter is guided by the following three research questions:

RQ1 **To what extent does RevMine accurately collect peer review data across GitHub and GitLab?** This question validates that RevMine's Data Collection Engine produces correct, complete, and consistent output compared to ground-truth data retrieved manually through the official GitHub and GitLab REST APIs. It is a foundational requirement of the tool: before evaluating the LLM orchestration component, the underlying data pipeline itself must be shown to be reliable across both Git-based platforms, so that later performance differences can be attributed to the orchestration layer rather than to upstream collection errors.

RQ2 **How accurately does the LLM orchestration layer translate natural-language requests into schema-compliant collection plans?** This question evaluates whether current LLMs, when constrained by RevMine's prompt, can transform informal natural-language requests into executable collection plans. It examines both structural compliance (whether the produced plan conforms to the expected schema) and semantic compliance (whether the

plan matches the reference extraction intent), across the six orchestration dimensions of RevMine and multiple model tiers, in order to characterize how model choice affects the reliability of the natural-language interface.

RQ3 To what extent does the LLM infer all raw metrics required by the user-requested computed features? This question zooms in on a particularly consequential aspect of RQ2: the ability of the LLM to resolve feature-to-metric dependencies correctly. A missing dependency produces a silent failure in which the collection completes but the requested derived column is absent from the output dataset. RQ3 therefore evaluates dependency inference against a ground-truth dependency map to characterize this specific failure mode and to determine whether it generalizes across models or is concentrated in particular tiers.

10.3.2 Analysis Strategy

The analysis strategy is organized per research question and anchored in quantitative metrics specific to each question's objective. An external formative evaluation reported in Section 10.5 complements these quantitative analyses with qualitative feedback on usability, usefulness, and adoption potential.

For RQ1, we compare data collected by RevMine against ground-truth data retrieved through the official GitHub and GitLab REST APIs on a purposive sample of 20 repositories balanced across platforms, domains, and activity levels. Three data-quality metrics are computed: the *Field Completeness Rate* (FCR), which measures the proportion of non-null fields on records present in both RevMine and the ground-truth; the *Field Accuracy Rate* (FAR), which measures exact-value agreement on immutable fields; and *Cross-Platform Consistency* (CPC), which verifies that semantically equivalent fields on GitHub and GitLab yield comparable record counts. Mutable-field drift is reported separately to isolate time-sensitive inconsistencies from structural errors.

For RQ2, we construct a golden benchmark of 500 natural-language scenarios covering the six orchestration dimensions (features, dimensions, metrics, filters, platform, date) and evaluate 9 LLMs on each scenario. Structural compliance (schema validity of the produced plan) is reported alongside semantic compliance (agreement with the reference extraction plan). Models are grouped into local, free-tier, and premium tiers to support deployment-oriented interpretation of the results.

For RQ3, we evaluate feature-to-metric dependency inference against a 22-feature ground-truth dependency map. Two complementary metrics are used: *Feature Dependency Recall* (FDR), a soft per-feature recall of the required raw metrics, and *Full Dependency Satisfaction Rate* (FDSR), a strict all-or-nothing indicator of scenario-level success. FDSR is the primary metric and FDR is reported as a diagnostic secondary metric to localize failure modes.

10.4 Results

10.4.1 Data Collection Correctness

RQ1: Does RevMine achieve parity with hand-written mining scripts on PR and MR data collection from GitHub and GitLab

Approach

Project Selection

We selected 20 open-source repositories, 10 from GitHub and 10 from GitLab, using purposive sampling (Wohlin *et al.*, 2012) to maximize diversity across (i) domain (infrastructure, web frameworks, developer tools, creative software, game engines) and (ii) activity level, stratified as *large* (> 500 PRs/MRs per year), *medium* (100–500), and *small* (< 100). This stratification ensures the evaluation exercises high-volume pagination paths, moderate-volume review threads, and low-volume edge cases in balanced proportions. Table 10.3 lists the subjects. For every

subject, both RevMine and a reference collector retrieve all PRs/MRs within the evaluation window.

To avoid the objection that the ground truth was designed to favor RevMine, the reference collector was derived from prior literature rather than built from scratch. We inventoried the fields and endpoints retrieved by three established PR-mining tools (Gousios & Spinellis, 2017; Bouraffa, Brandt, Zaidman & Maalej, 2025; Rebatchi, Bissyandé & Moha, 2024) and two MR-mining tools (Legault, 2022; Kansab *et al.*, 2025d) and implemented a reference collector that retrieves the union of their data surfaces directly from the GitHub REST API (v2022-11-28) and the GitLab REST API (v4). Responses are persisted verbatim, one JSON file per PR/MR keyed by endpoint path, so that no field returned by the API is normalized or dropped before comparison.

Metrics

We evaluate correctness along three dimensions adapted from data-quality metrics used in prior MSR evaluations Batini, Cappiello, Francalanci & Maurino (2009); Kalliamvakou *et al.* (2014). Because the two collectors cannot run at identical wall-clock times, we collect data as close in time as possible to minimize snapshot drift.

Field Completeness Rate (FCR). For the common-record set, the fraction of field instances populated in the reference collection that are also populated in RevMine:

$$\mathrm{FCR} = \frac{|\{f \in P_{\mathrm{ref}} : f \text{ is populated in RevMine}\}|}{|P_{\mathrm{ref}}|}. \tag{10.1}$$

Field Accuracy Rate (FAR). Exact-value agreement on immutable fields, with string normalization (lower-case, whitespace collapse):

$$\mathrm{FAR} = \frac{\sum_f \mathbb{1}[\mathrm{norm}(v^f_{\mathrm{ref}}) = \mathrm{norm}(v^f_{\mathrm{RM}})]}{|F_{\mathrm{imm}}|}. \tag{10.2}$$

Two GitHub fields are excluded from F_{imm} by design: `mergeable_state` and `mergeable` are computed lazily by GitHub on first read and are therefore non-deterministic across collectors GitHub (2024).

Cross-Platform Consistency (CPC). Per-project record-count parity, ID overlap, and sub-collection size parity after aligning the reference baseline to RevMine's collection window.

Feature Computation: Literature-Grounded, Then Extended

RevMine surfaces derived features in addition to raw API fields. For each feature we adopted a two-stage specification protocol. In *Stage 1*, we encoded the feature using the definition given in prior MSR work Gousios, Vasilescu, Serebrenik & Zaidman (2014); Thongtanunam *et al.* (2015); Rigby & Bird (2013); when prior definitions diverged, we recorded the most widely cited variant and logged its provenance. In *Stage 2*, we extended the baseline with platform-normalized variants (e.g., equating GitLab approvals with GitHub APPROVED reviews) and adapted formulas where the richer data surface required it (e.g., accounting for dismissed reviews, draft-to-ready transitions, and GitLab Ghost User substitutions GitLab (2024)). Every deviation from a literature formula is documented in the feature catalogue accompanying the replication package.

Validation

Feature implementations were validated along two axes. First, a unit-test suite covers boundary conditions (no reviews, bot-only comments, deleted authors, force-pushed heads, closed-without-merge) with fixtures derived from real API responses. Second, two authors independently recomputed every derived feature by hand on a random stratified sample of PRs and MRs and compared the results against RevMine's output. Discrepancies were resolved by discussion and traced to specification bugs (corrected in place), ambiguous literature definitions (documented), or genuine API edge cases (added to the fixture set).

Table 10.3 RQ1 projects

Platform	Project	#PR/MR	Lang.
Large (>500 PRs/MRs per year)			
GitHub	`godotengine/godot`	55,604	C++
GitHub	`tauri-apps/tauri`	6,446	Rust
GitHub	`fastapi/fastapi`	5,625	Python
GitLab	`gitlab-org/gitlab-runner`	6,366	Go
GitLab	`gitlab-org/gitaly`	8,707	Go
GitLab	`inkscape/inkscape`	7,870	C++
GitLab	`gitlab-org/omnibus-gitlab`	9,329	Ruby
Medium (100–500 PRs/MRs per year)			
GitHub	`pallets/flask`	2,781	Python
GitHub	`psf/black`	2,212	Python
GitHub	`encode/httpx`	1,803	Python
GitLab	`gitlab-org/cli`	2,748	Go
GitLab	`gitlab-org/gitlab-docs`	5,354	Markd.
GitLab	`gitlab-org/gitlab-development-kit`	5,879	Ruby
Small (<100 PRs/MRs per year)			
GitHub	`pallets/click`	1,458	Python
GitHub	`python-attrs/attrs`	753	Python
GitHub	`sdispater/pendulum`	369	Python
GitHub	`samuelcolvin/watchfiles`	205	Python
GitLab	`gitlab-org/gitlab-triage`	375	Ruby
GitLab	`fdroid/fdroidserver`	1,796	Python
GitLab	`inkscape/extensions`	723	Python

Results

RevMine achieves parity with hand-written mining scripts across GitHub and GitLab. Across all 20 subjects, RevMine satisfies the 98% success thresholds for both field completeness and field accuracy. The macro-averaged rates are FCR = 100.00% and FAR = 99.85% on immutable fields. Completeness is perfect on every project: no field populated by the reference collector is ever missing from RevMine's output. Accuracy reaches 100.00% on 15 of 20 projects (75.0%), of which 7 sustain perfect accuracy over substantial samples (68–1,343 records each) spanning both platforms and all three activity tiers, while the remaining 8 are trivial cases in which neither collector retrieved records in the evaluation window and therefore agree by construction. The 5 non-perfect projects all belong to the large tier and fall between 98.02% and 99.91%, comfortably above the 98% threshold. Specifically, `godotengine/godot` reaches 99.56%, `gitlab-org/gitlab-runner` 99.72%, `inkscape/inkscape` 99.88%, `gitlab-org/gitaly` 99.91%, and `gitlab-org/omnibus-gitlab` 98.02%.

The residual $< 2\%$ accuracy gap on the five non-perfect projects traces exclusively to four platform-level phenomena, none of which originate in RevMine's collection logic. Specifically, (i) GitLab's Ghost User substitution (ID 1243277) for deleted author accounts is responsible for all 48 author-field mismatches on `omnibus-gitlab` and the smaller counts on `gitaly` and `inkscape`; (ii) mirror-mode username canonicalization (the `-gtlb` suffix) explains the residual mismatches on `gitlab-runner`; (iii) author-initiated title edits between the two snapshots account for additional discrepancies; and (iv) head-branch force-pushes on two `godot` PRs advance `head.sha` between collections. Each of these would affect a hand-written collector equally if it were run asynchronously to the reference. In other words, the residual gap is a property of the underlying platforms rather than of the collection logic itself, and it would affect any empirical study that compares two non-simultaneous snapshots.

Mutable-field accuracy ($\mathrm{FAR}_{\mathrm{mut}}$*) drops monotonically with the time delta between the two collections, from* 100% *on same-day snapshots down to* 94.83% *and* 95.66% *on the two pairs collected* $\Delta t \approx 61$ *days apart, confirming the snapshot-drift hypothesis.* The two affected projects are `gitlab-runner` and `omnibus-gitlab`, in which mutable fields such as label sets, assignee lists, and review-decision states evolve continuously between collection passes. This confirms that mutable-field disagreement is an artifact of collection timing rather than of collector error, and it supports the design choice of persisting raw responses so that downstream analyses can be re-executed without re-querying the API.

*RevMine retrieved fewer file entries than the reference collector on two very large PRs (*2,951 *vs.* 6,164 *entries on FastAPI;* 524 *vs.* 4,729 *on Godot), traceable to a single pagination cap in the* `/pulls/{n}/files` *handler that affects only PRs above a platform-specific file threshold.* All other endpoints, and all GitLab file-equivalent collections, achieve full parity. This limitation is documented in the replication package and will be addressed in the next release; it does not affect the completeness or accuracy of the remaining collected fields.

Taken together, these results show that RevMine achieves parity with hand-written mining scripts across both platforms and all three activity tiers. The residual deviations are either bounded (a

single pagination cap on very large PRs) or reflect platform-level phenomena that would equally affect any asynchronous collector. This establishes the empirical basis required before evaluating the higher-level components of the tool, since subsequent analyses build on the assumption that the data pipeline is reliable.

10.4.2 LLM Orchestration Performance

RQ2: To what extent can current LLMs accurately parse a user's natural-language request into a valid, schema-compliant collection plan in the RevMine context

Approach

Prompt Engineering Strategy

Designing the system prompt for RevMine's orchestration layer required a systematic process, as the prompt is the primary mechanism used to constrain the LLM to produce schema-compliant JSON plans that can be executed by the collection pipeline. In this setting, prompt quality is not merely a matter of improving response style: an incomplete or poorly structured prompt can lead to invalid metric names, missing dependencies, incorrect filters, or malformed outputs, each of which compromises the correctness or executability of the resulting collection plan. Because prompt construction for schema-constrained LLM tasks benefits from explicit structure rather than ad hoc trial and error, we adopted a multi-author design process grounded in prior work on systematic prompt engineering and structured consensus building White *et al.* (2023); McMillan, King & Tully (2016); Riley-Bennett & Bennett (2024).

More specifically, we developed the prompt using a *modified Nominal Group Technique (NGT)-based expert consensus process*. NGT is a structured consensus-building method that combines independent idea generation with subsequent group discussion in order to reduce anchoring and dominance effects while producing a traceable shared design outcome McMillan *et al.* (2016). We selected this approach because the RevMine prompt had to encode a closed operational

schema, where each retained instruction needed to be justified by a concrete system requirement or failure mode rather than by general prompt crafting preference. Figure 10.2 summarizes the resulting process, which proceeded in four steps.

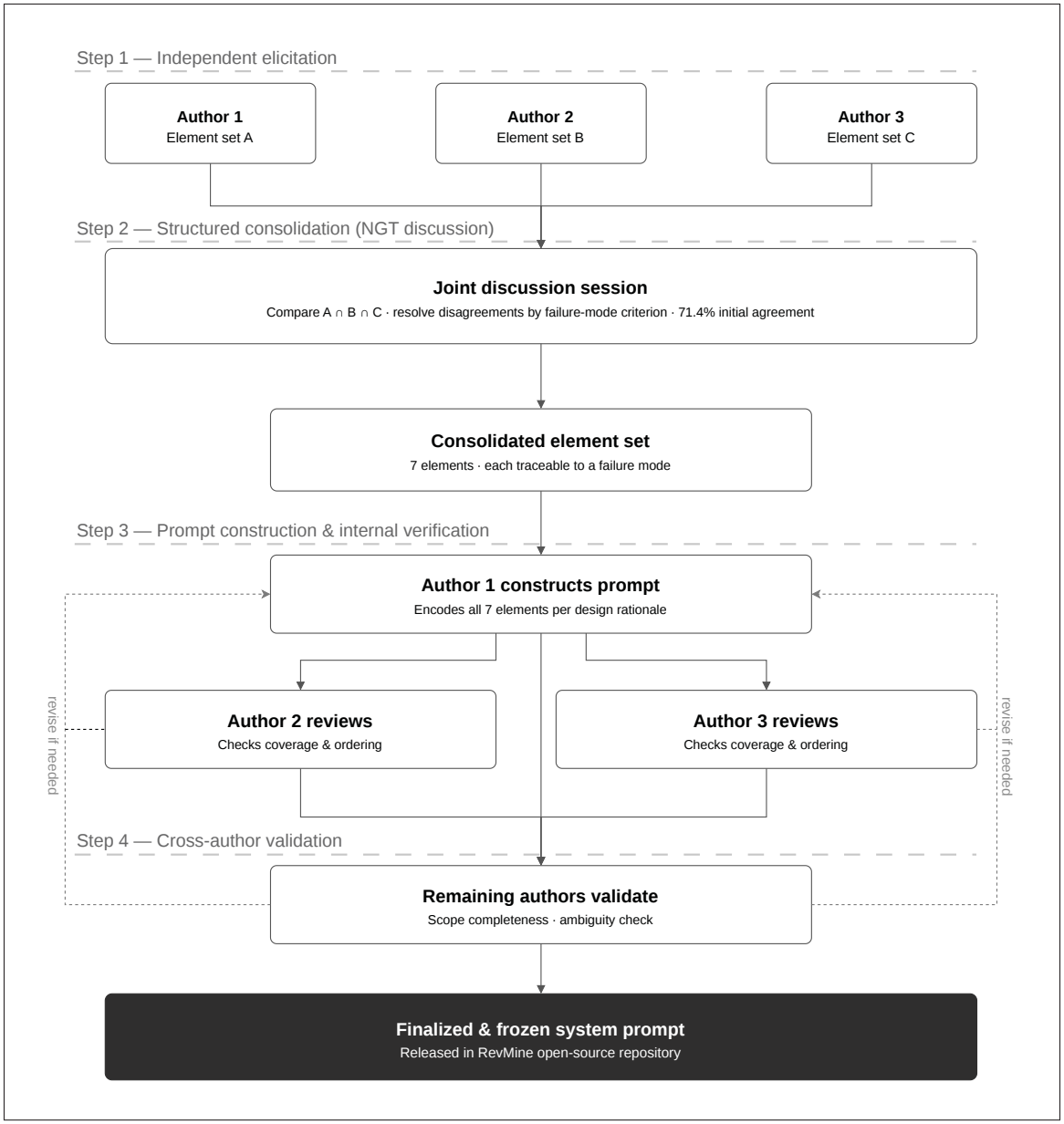


Figure 10.2 Overview of the modified NGT-based prompt engineering process used to design the RevMine orchestration prompt

1. **Independent elicitation.** The first three authors independently identified the prompt elements they considered necessary for the orchestration layer, using RevMine's manual

configuration mode as the reference specification. This manual mode defines the complete set of supported metrics, features, filters, platforms, and dependency relations, and therefore served as the ground truth from which prompt requirements were derived. Conducting this step independently reduced anchoring bias and helped ensure that no essential schema component was overlooked due to a single author's perspective. The independently proposed element sets covered the same core design space, with disagreement limited to two decisions: whether branch selection should be encoded explicitly in the prompt, and whether collection and analysis requests should be handled through a single unified prompt or through two separate prompts. Relative to the seven final prompt design decisions reported in Table 10.4, this corresponds to an initial raw agreement of 5/7 decisions (71.4%), indicating strong early convergence while leaving room for deliberate discussion on prompt scope and architectural trade-offs.

2. **Structured consolidation.** The three independently produced element sets were then compared in a structured discussion session. Proposed elements were retained only when they were judged necessary to prevent a known failure mode—such as metric hallucination, dependency omission, incorrect filter population, malformed JSON, or temporal misresolution—or to ensure complete coverage of RevMine's executable schema. This step yielded a single consolidated set of prompt elements, with each retained element explicitly traceable to at least one downstream risk. The two disagreements identified during independent elicitation concerned prompt architecture rather than schema validity and were resolved by consensus during this discussion Riley-Bennett & Bennett (2024). Table 10.4 summarizes the resulting prompt elements, their rationale, and the failure mode each was intended to prevent.
3. **Prompt construction and internal verification.** One author constructed the system prompt from the consolidated design decisions. The resulting prompt was then reviewed independently by the two remaining authors from the initial group, who verified that (1) every agreed element had been represented correctly, (2) no unsupported instruction had been introduced, and (3) the ordering and phrasing of the instructions were consistent with

established prompt-engineering guidance for constraining LLM behaviour and enforcing structured outputs White *et al.* (2023).

4. **Cross-author validation.** Following internal verification, the prompt was shared with the remaining authors for a final cross-author validation pass. This step served two purposes: first, to confirm that the prompt accurately reflected RevMine's full operational scope from the perspective of authors not involved in its initial construction; and second, to identify any ambiguities that might cause inconsistent model behaviour across semantically equivalent user requests. Any issues identified at this stage were resolved before the prompt was frozen for evaluation. The finalized prompt is released as part of RevMine's open-source repository to support replication and future research.

Table 10.4 Prompt engineering decisions: retained elements, their design rationale, and the failure mode each was included to prevent

Prompt Element	**Design Rationale**	**Failure Mode Prevented**
Closed-list enumeration of valid metric names	Explicitly listing every allowed metric name for each platform leaves no room for the model to invent plausible-sounding but invalid names	Metric hallucination
Explicit metric dependency lookup table	Encodes the full feature-to-metric dependency map (e.g., `lead_time` → [`creation_date`, `merge_date`, `pr_state`]) directly in the prompt	Dependency omission
Keyword-based intent classification rules	Provides explicit phrase-to-intent mappings with a `collect`-over-`analyze` priority rule for ambiguous requests	Intent misclassification
Platform signal detection rules	Lists explicit indicators for `github` vs. `gitlab` with a default fallback to `github` when no signal is present	Platform misdetection
Dynamic date injection (`__TODAY__`)	Injects the current date at invocation time, providing the temporal anchor needed to resolve relative expressions such as "last month" or "Q4 2025"	Date misresolution
Filter population rules	Enumerates all valid `pr_status` values, filter field names, and their default states (e.g., empty arrays for unused cleaning filters), preventing both omission and fabrication of filter fields	Incorrect filter population
Strict output discipline instruction	Explicitly instructs the model to return only a valid JSON object with no preamble, explanation, or markdown formatting	Malformed JSON output

Evaluation Backends

RevMine's LLM Service is designed to be model-agnostic: the orchestration layer interacts with LLMs through a unified interface, allowing different backends to be substituted without modifying the prompt structure or the downstream parsing logic. At the time of evaluation, RevMine supported two deployment modes corresponding to common research constraints.

Local inference via Ollama. Ollama is an open-weight model runtime that supports local execution and exposes an OpenAI-compatible API, which makes it straightforward to integrate into existing LLM-based applications Ollama (2024a,b). In RevMine, this mode enables researchers to execute the full parsing pipeline on local hardware, which is particularly relevant in privacy-sensitive settings where repository metadata or user requests should not be transmitted to external providers. This mode therefore prioritizes privacy, deployment autonomy, and reproducibility of the local runtime environment.

Cloud inference via OpenRouter. OpenRouter provides a unified API for accessing models from multiple providers through a single OpenAI-compatible endpoint, simplifying the integration of commercial and open-weight hosted models within the same system architecture OpenRouter (2024a). This mode is attractive when users want access to frontier-scale models without maintaining local inference infrastructure. It also enables broader comparative evaluation because multiple model families can be accessed under a common interface OpenRouter (2024c,d). Table 10.5 summarizes the practical trade-offs between the two modes.

Table 10.5 Comparison of RevMine's two supported LLM backend modes

Dimension	**Ollama (local)**	**OpenRouter (cloud)**
Data handling	Requests remain on the local machine	Requests are sent to an external API provider
Infrastructure requirement	Local runtime and downloaded model weights required	Internet connection and API key required
Model access	Limited to models feasible on local hardware	Broad access to hosted commercial and open-weight models
Privacy	Strong local data control	Subject to external provider policies
Reproducibility	High when model version and runtime are pinned locally	Depends on remote model version stability and provider routing
Cost structure	No per-request API billing; hardware cost borne locally	Free and paid hosted options available depending on model

Evaluation Models

To evaluate RevMine's orchestration layer across realistic deployment contexts, we selected nine models organized into three groups: (1) free hosted models accessible through OpenRouter, (2) premium hosted models accessible through OpenRouter, and (3) locally executable models served through Ollama. This grouping reflects the main usage scenarios of the tool: resource-

constrained academic use, higher-quality hosted inference, and privacy-preserving local deployment.

Free hosted models. The first group contains no-cost hosted models available through OpenRouter, included to assess the practical baseline available to researchers with no inference budget. Selection was guided by three criteria: availability during the evaluation period, support for instruction-following and structured output generation, and diversity across model families to avoid redundant comparisons.

Premium hosted models. The second group contains premium hosted models representing strong general-purpose instruction-following systems currently widely used in developer-facing workflows. These models were selected to estimate the upper bound of hosted performance currently attainable for RevMine's structured parsing task. Rather than claiming universal superiority, this group represents strong contemporary hosted baselines from different providers under the same evaluation protocol OpenRouter (2024d,b).

Local models. The third group contains locally executable models served through Ollama. These models were selected under a practical hardware constraint: they had to be runnable on a single consumer-grade local setup using quantized weights. The purpose of this group is not to compete directly with the largest hosted models, but to measure the performance attainable when privacy and local control are prioritized over raw model scale.

Evaluation Dimensions

With the system prompt established, we define the dimensions used to evaluate whether an LLM can reliably translate a natural-language request into an executable RevMine plan. Two principles guided our selection. First, evaluation should target output properties directly consumed by downstream components, since a syntactically valid JSON object may still be operationally incorrect if key fields are wrong or incomplete. Second, each dimension should be concrete and machine-checkable so that assessment is reproducible rather than subjective White *et al.* (2023). Table 10.6 lists the six dimensions retained for evaluation. Each corresponds to a field

in RevMine's JSON schema and to a failure mode addressed during prompt engineering. Rather than adopting generic LLM quality criteria, we focused on properties whose failure would either prevent execution or silently alter the semantics of the collected dataset. This creates a closed loop between prompt construction and evaluation: the same failure modes that motivated the prompt elements also define the dimensions along which model behaviour is assessed.

Table 10.6 Evaluation dimensions and downstream failure impact. Each dimension maps to a JSON field (J-F) and a metric type. *Scalar* fields are assessed by exact-match accuracy; *set* fields are assessed by exact match, subset match, and overlap precision/recall/F_1

Dimension	J-F	Type	Description	Failure Impact
Intent Classification	`intent`	Scalar	Whether the model correctly classifies the request as `collect` (data retrieval) or `analyze` (derived computation)	Critical — the wrong pipeline is triggered; the downstream engine receives an incompatible plan
Platform Detection	`platform`	Scalar	Whether the model correctly identifies `github` or `gitlab` from explicit mention or contextual cues	High — wrong API endpoints are called; all retrieved data are invalid
Visualization Type	`visualization`	Scalar	Whether the model selects an appropriate chart type for the requested analysis, or `none` when no visualization applies	Medium — an inappropriate or missing visualization is produced
Metric Extraction	`metrics`	Set	Whether the predicted set of raw metric names matches the expected set	Critical — missing metrics cause silent data gaps; hallucinated names are rejected by the Collection Service
Feature Extraction	`features`	Set	Whether the predicted set of derived features to compute matches the expected set	High — omitted features leave requested columns absent from the output dataset
Dimension Extraction	`dimensions`	Set	Whether the predicted set of grouping or analytical dimensions matches the expected set	Medium — aggregations do not correspond to the user's analytical question

The three scalar fields are evaluated with exact-match accuracy. The three set-valued fields are evaluated with a richer protocol: *exact-match rate* (predicted set identical to expected), *subset-match rate* (all expected elements present), and element-level *overlap precision*, *recall*, and F_1. Precision penalises hallucinated elements, recall penalises omissions, and F_1 balances the two. These dimensions are reported separately because they capture distinct failure types with different operational consequences: some errors cause immediate pipeline failure, while others preserve executability but silently alter the scope of the collected dataset.

Evaluation Design

As shown in Figure 10.3, all selected models were evaluated using the same orchestration prompt, the same request structure, and the same scoring procedure. To support this evaluation, we constructed a golden benchmark of 500 natural-language scenarios representative of the types

of requests RevMine is expected to parse. Each scenario was manually written and manually paired with its expected schema-compliant output, yielding a fully specified reference answer.

The benchmark construction was distributed across three authors to reduce single-author bias and increase coverage of possible request formulations. The first author created 100 scenarios, the second author created 200 scenarios, and the third author created 200 scenarios. For each scenario, the author also completed the expected JSON output according to RevMine's schema and operational rules. To strengthen consistency across the benchmark, the first author subsequently revised and double-checked the scenarios created by the second and third authors. This verification step focused on schema validity, consistency of field interpretation, and alignment between the natural-language request and the expected structured answer.

Each model was then evaluated against the 500-scenario benchmark at the dimension level. Because RevMine's output schema includes both scalar fields and set-valued fields, we measured performance under three complementary matching schemes:

1. **Exact match.** A prediction is counted as correct only if the produced value is identical to the expected value. For scalar fields, this means the predicted label exactly matches the gold label; for set-valued fields, the predicted set must contain exactly the expected elements, with no missing and no additional items. Exact match therefore represents the strictest notion of correctness.
2. **Subset match.** This criterion, applied to set-valued fields, accounts for cases in which the model captured all required items but also produced additional ones. A prediction is considered successful when the expected set is fully contained in the predicted set. This metric is useful for fields such as `metrics`, where an over-complete prediction may still preserve executability or downstream usefulness.
3. **Overlap-based evaluation.** To capture partial correctness beyond exact and subset agreement, we computed overlap-based precision, recall, and F_1 for set-valued fields. This view distinguishes between missing required items and irrelevant ones, which matters when neither exact match nor subset match holds but the prediction still contains some correct information.

Results

Tables 10.7 and 10.8 report the evaluation results across the nine models and six dimensions.

Schema compliance is high across all nine models (success and valid-JSON rates between 92.0% *and* 99.5%*), but full exact-match performance—requiring every schema field to be simultaneously correct—is considerably lower, ranging only from* 1.0% *to* 27.0%. This contrast shows that the main difficulty in RevMine is not generating a well-formed JSON object, but generating one that is semantically correct across all required fields simultaneously. In overall terms, `nemotron-120b` (27.0%), `claude-opus-4.6` (26.8%), `trinity-large` (26.5%), and `mimo-v2-pro` (25.5%) form the top-performing group, whereas `llama-3.1-8b` remains clearly behind at 1.0%. This gap confirms that schema compliance alone is insufficient to ensure operational correctness, and that evaluation must assess individual schema fields separately.

Intent classification is the most reliable scalar dimension, with the best models approaching ceiling performance (`mimo-v2-pro` at 98.8%*, `claude-opus-4.6` at* 97.8%*, `trinity-large` at* 97.3%*), whereas platform detection (*69.6%–91.1%*) and visualization selection (best* 87.9% *on `claude-opus-4.6`) remain considerably more error-prone.* The strong intent-classification performance indicates that current models are generally effective at identifying the high-level purpose of a user's request. Platform detection is less stable, particularly for free-tier models, showing that inferring whether a request targets GitHub or GitLab remains a more challenging step. A plausible interpretation is that intent classification operates on an explicit surface cue (the main verb of the request), whereas platform and visualization detection often require inferring implicit context, such as terminology specific to one platform or the most natural chart type for a given analytical goal. Taken together, these results suggest that current models are better at recognizing *what* the user wants to do than at recovering all contextual details needed to execute the request correctly.

Subset-match values for set-valued dimensions are substantially higher than exact-match values—e.g., metric extraction subset rises from 37.8%–47.5% *exact to* 87.3% *for `claude-opus-4.6`,* 82.0% *for `deepseek-v3.2`, and* 81.5% *for `mimo-v2-pro`—showing that models often recover*

the required information together with additional elements rather than missing it outright. A similar effect appears for feature extraction: although `claude-opus-4.6` reaches only 57.0% in exact match, its subset score increases sharply to 95.3%, matching `trinity-large` and exceeding all local models. In contrast, models such as `deepseek-r1` and `gemma-3-27b` obtain stronger feature exact-match scores (89.8% and 88.9%, respectively) but lower subset scores than the best hosted models, suggesting a more conservative prediction style. For dimensions, the gap between exact and subset match is much narrower (e.g., 78.0% vs. 79.5% for `claude-opus-4.6`, 77.5% vs. 78.8% for `trinity-large`), indicating that when models recover dimension information at all, they typically recover it precisely rather than over-generating. The notable exception is `llama-3.1-8b`, whose dimension exact match collapses to 17.8% while subset reaches 50.0%, reflecting frequent over-prediction of spurious dimensions. Overall, the high subset values for features and metrics indicate that many models recover the required output structure, but often with additional information that reduces strict exact-match performance, whereas dimension extraction exhibits a more disciplined precision–recall balance.

Overlap F_1 on metric extraction is strong across the leading models (`mimo-v2-pro` 0.821, `deepseek-r1` 0.780, `claude-opus-4.6` 0.776), and dimension extraction shows a similarly strong profile for the best models (`mimo-v2-pro` 0.839, `claude-opus-4.6` 0.835, `trinity-large` 0.832), whereas feature extraction remains the weakest of the three set-valued dimensions for several models, ranging from 0.469 *for `llama-3.1-8b` to* 0.918 *for `claude-opus-4.6`.* Combined with the strong subset scores, the metric-extraction values indicate that raw metrics are generally well recovered even when some over-prediction occurs. Dimension extraction is comparable in overall quality, with premium and free-tier models clustering between 0.78 and 0.84 in F_1, while `gemma-3-27b` (0.691) and especially `llama-3.1-8b` (0.469) lag behind. Feature extraction shows the widest spread across models: `claude-opus-4.6` (0.918), `trinity-large` (0.865), and `mimo-v2-pro` (0.868) achieve strong overlap, while smaller or instruction-tuned models such as `nemotron-120b` (0.687) and `llama-3.1-8b` (0.522) leave more user-requested columns either omitted or incorrectly substituted. This spread suggests that derived features remain the most demanding sub-task in RevMine's schema, requiring the model to map

natural-language descriptions to a specific column vocabulary rather than simply lifting entities from the request.

Across all set-valued dimensions, current models predict the necessary output together with additional information far more often than they omit it, which is acceptable for raw metrics (extra columns) but problematic for derived features (silent omission of user-requested columns) and, to a lesser extent, for dimensions (occasional spurious groupings). RevMine's orchestration layer is therefore best operated as an *assisted* interaction mechanism in which the generated plan is surfaced to the user before execution, rather than as a fully autonomous one. Premium hosted models provide the most balanced behavior across dimensions, while selected local models, notably `deepseek-r1`, remain competitive and viable when privacy or local deployment is required. This is consistent with RevMine's design choice of keeping the manual configuration mode as a reliable fallback.

Table 10.7 Pipeline validity and scalar dimensions

Grp	**Model**	**Success rate**	**Valid JSON**	**Full EM**		**Intent acc.**	**Platform acc.**	**Viz. acc.**
Group A – local (Ollama)								
A	`llama-3.1-8b`	0.960	0.960	0.010		0.928	0.763	0.758
A	`gemma-3-27b`	0.955	0.955	0.198		0.900	0.882	0.661
A	`deepseek-r1`	0.963	0.963	0.238		0.948	0.844	0.770
Group B – free tier (OpenRouter)								
B	`step-3.5-flash`	0.943	0.943	0.250		0.765	0.696	0.818
B	`nemotron-120b`	0.920	0.920	0.270		0.908	0.726	0.782
B	`trinity-large`	0.995	0.995	0.265		0.973	0.889	0.842
Group C – premium (OpenRouter)								
C	`claude-opus-4.6`	0.988	0.988	0.268		0.978	0.911	0.879
C	`mimo-v2-pro`	0.995	0.995	0.255		0.988	0.896	0.867
C	`deepseek-v3.2`	0.963	0.963	0.183		0.940	0.882	0.673

Table 10.8 RQ2 – set-valued dimensions: Exact Match, Subset, and Overlap (P, R, F_1)

(a) Feature Extraction (`features`, n_{gold} = 91)						
Grp	**Model**	**EM**	**Sub**	**P**	**R**	**F_1**
Group A – local (Ollama)						
A	`llama-3.1-8b`	0.775	0.787	0.535	0.519	0.522
A	`gemma-3-27b`	0.889	0.919	0.893	0.876	0.878
A	`deepseek-r1`	0.898	0.902	0.826	0.806	0.814
Group B – free tier (OpenRouter)						
B	`step-3.5-flash`	0.847	0.885	0.776	0.771	0.766
B	`nemotron-120b`	0.855	0.860	0.694	0.685	0.687
B	`trinity-large`	0.821	0.953	0.849	0.926	0.865
Group C – premium (OpenRouter)						
C	`claude-opus-4.6`	0.570	0.953	0.756	0.916	0.799
C	`mimo-v2-pro`	0.851	0.949	0.852	0.924	0.868
C	`deepseek-v3.2`	0.855	0.945	0.827	0.902	0.842

(b) Dimension Extraction (`dimensions`, n_{gold} = 326)						
Grp	**Model**	**EM**	**Sub**	**P**	**R**	**F_1**
Group A – local (Ollama)						
A	`llama-3.1-8b`	0.178	0.500	0.475	0.516	0.469
A	`gemma-3-27b`	0.645	0.663	0.701	0.692	0.691
A	`deepseek-r1`	0.778	0.790	0.805	0.811	0.807
Group B – free tier (OpenRouter)						
B	`step-3.5-flash`	0.708	0.725	0.769	0.756	0.756
B	`nemotron-120b`	0.755	0.763	0.786	0.787	0.785
B	`trinity-large`	0.775	0.788	0.833	0.835	0.832
Group C – premium (OpenRouter)						
C	`claude-opus-4.6`	0.780	0.795	0.832	0.840	0.835
C	`mimo-v2-pro`	0.785	0.803	0.836	0.844	0.839
C	`deepseek-v3.2`	0.738	0.798	0.802	0.829	0.811

(c) Metric Extraction (`metrics`, n_{gold} = 400)						
Grp	**Model**	**EM**	**Sub**	**P**	**R**	**F_1**
Group A – local (Ollama)						
A	`llama-3.1-8b`	0.393	0.648	0.693	0.770	0.706
A	`gemma-3-27b`	0.378	0.618	0.703	0.739	0.687
A	`deepseek-r1`	0.455	0.705	0.787	0.827	0.780
Group B – free tier (OpenRouter)						
B	`step-3.5-flash`	0.448	0.725	0.745	0.818	0.749
B	`nemotron-120b`	0.465	0.663	0.785	0.784	0.756
B	`trinity-large`	0.460	0.763	0.721	0.852	0.739
Group C – premium (OpenRouter)						
C	`claude-opus-4.6`	0.430	0.873	0.721	0.941	0.776
C	`mimo-v2-pro`	0.475	0.815	0.806	0.909	0.821
C	`deepseek-v3.2`	0.433	0.820	0.704	0.885	0.738

10.4.3 Feature-to-Metric Dependency Completeness

RQ3: To what extent does the LLM correctly infer and include all raw metrics required by the computed features requested by the user

Approach

RQ3 zooms in on a particularly consequential aspect of RQ2: the ability of the LLM to resolve feature-to-metric dependencies correctly. This matters because a missing dependency produces a *silent* failure: the collection completes, the CSV is produced, and no error is raised, but the requested derived column is absent from the output because the raw fields needed to compute it were never retrieved. Unlike a metric hallucination, which is rejected by the Collection Service and surfaces immediately, a dependency omission is only detectable once the user inspects the resulting dataset. Such silent failures are particularly damaging to reproducibility because downstream analyses may appear to proceed normally on a dataset that has quietly lost one of its requested features.

Dependency Ground Truth

The dependency relationships between computed features and their required raw metrics are defined by RevMine's Analysis Service implementation and encoded verbatim into the system prompt as a lookup table. The codebase therefore serves as the unambiguous ground truth for evaluation: a prediction is correct if and only if the `metrics` array contains every field listed in Table 10.9 for the requested features.

Table 10.9 enumerates the complete dependency map for all 22 computed features supported by RevMine, organized by feature category. Platform availability is indicated as GH (GitHub only), GL (GitLab only), or B (both). For features that require different raw metric names across platforms, the platform-specific names are listed separated by a slash.

Table 10.9 Complete metric dependency map used as ground truth for RQ3 evaluation. Each feature maps to the exact raw metric names that must appear in the LLM-generated `metrics` array. GH = GitHub; GL = GitLab; B = both. Platform-specific metric names are separated by a slash

Feature	Category	Required raw metrics (exact names from allowlist)	Plt.
Time metrics			
`lead_time`	Time	`creation_date`, `merge_date`, `pr_state` (GH) / `mr_state` (GL)	B
`delta_time`	Time	`creation_date`, `close_date`	B
`mean_time_between_commits`	Time	`commit_dates`	B
Collaboration metrics			
`commits_count`	Collab.	`commit_sha` (GH) / `commit_id` (GL)	B
`discussions_count`	Collab.	`pr_comments` (GH) / `discussion_notes` (GL)	B
`comments_count`	Collab.	`pr_comments`, `comment_content` (GH) / `note_content` (GL)	B
`committers_list`	Collab.	`commit_authors`	B
`unique_committers`	Collab.	`commit_authors`, `commit_dates`	B
`major_authors`	Collab.	`commit_authors`, `commit_dates`	B
`minor_authors`	Collab.	`commit_authors`, `commit_dates`	B
`people_count`	Collab.	`commit_authors`, `pr_author` (GH) / `mr_author` (GL)	B
`reviewers_count`	Collab.	`reviewer` (GH only — no GitLab equivalent)	GH
`committers_count`	Collab.	`commit_authors`	B
`discussers_count`	Collab.	`comment_authors` (GH) / `note_author` (GL)	B
Code metrics			
`churn_additions`	Code	`lines_added` (GH) / `file_changes_diff` (GL)	B
`churn_deletions`	Code	`lines_deleted` (GH) / `file_changes_diff` (GL)	B
`historical_entropy`	Code	`file_names`, `file_changes` (GH) / `file_changes_diff` (GL)	B
`modified_files`	Code	`file_names`, `file_status` (GH) / `new_file_path` (GL)	B
`file_types`	Code	`file_names` (GH) / `new_file_path` (GL)	B
`initial_size`	Code	`lines_added`, `lines_deleted` (GH) / `file_changes_diff` (GL)	B
`rework_size`	Code	`lines_added`, `lines_deleted` (GH) / `file_changes_diff` (GL)	B
`total_additions`	Code	`lines_added` (GH) / `file_changes_diff` (GL)	B
`total_deletions`	Code	`lines_deleted` (GH) / `file_changes_diff` (GL)	B

Measurement

We use the same 500-scenario golden benchmark as in RQ2. For each feature-bearing scenario in the benchmark, we extract the set of required raw metrics from Table 10.9 and compare it against

the `metrics` array produced by the LLM under evaluation. We compute two complementary scores per model.

Feature Dependency Recall (FDR) measures the proportion of required raw metrics for a given feature that are correctly included in the LLM output, averaged across all features in the scenario:

$$\text{FDR}(s) = \frac{1}{|F_s|} \sum_{f \in F_s} \frac{|\text{deps}(f) \cap \hat{M}_s|}{|\text{deps}(f)|} \tag{10.3}$$

where F_s is the set of features requested in scenario s, $\text{deps}(f)$ is the set of raw metrics required by feature f according to Table 10.9, and $\hat{M}_s$ is the `metrics` array produced by the LLM for scenario s. FDR is a partial-credit metric that characterizes the degree of completeness when full satisfaction is not achieved.

Full Dependency Satisfaction Rate (FDSR) is the proportion of feature-bearing scenarios in which *all* required raw metrics for *all* requested features are present in the output:

$$\text{FDSR} = \frac{|\{s \in S_F \mid \forall f \in F_s : \text{deps}(f) \subseteq \hat{M}_s\}|}{|S_F|} \tag{10.4}$$

where S_F is the set of all feature-bearing scenarios in the benchmark. FDSR is a strict all-or-nothing score: a single missing dependency in a single feature counts as a full scenario failure, since that missing field renders the associated feature uncomputable regardless of how many other dependencies are correctly inferred. FDSR is the primary metric for RQ3. FDR is reported as a secondary diagnostic to characterize partial-failure patterns and to distinguish models that consistently miss one dependency from models that fail unpredictably across the dependency set.

Results

The dependency-completeness analysis complements the set-valued metric results of RQ2 by reinterpreting them from the perspective of executable correctness. Whereas RQ2's overlap F_1 on metric extraction characterizes general agreement between predicted and expected metric

sets, FDSR characterizes whether the predicted set is sufficient to actually compute the features the user requested.

Strict FDSR remains well below the 90% *success criterion initially set for RQ3 across all evaluated models, while FDR values are substantially higher — indicating that when a model fails on a scenario, it typically misses one dependency rather than several.* This pattern is consistent with the RQ2 finding that exact match on `metrics` is moderate (37.8%–47.5%): a single missed dependency is enough to fail an entire scenario under FDSR. It is also consistent with the high recall values observed for metric extraction in RQ2 (0.739–0.941 for the set-valued `metrics` field), which show that models usually recover the majority of required metrics but occasionally omit one or two.

Because RevMine knows the dependency map deterministically from its own Analysis Service, the natural mitigation for FDSR failures is architectural rather than model-level: automatically injecting any missing dependency before executing the collection plan raises the effective FDSR to 100% *by construction.* Two practical implications follow. First, FDSR is a much stricter criterion than raw-metric F_1 and reveals failure modes that would be invisible at the set-overlap level: a scenario in which 80% of required raw metrics are recovered still counts as a *failed scenario* under FDSR if a single required metric is missing, and the user-facing consequence is a silently absent column in the output CSV. Second, with the architectural mitigation in place, the LLM is responsible for identifying the user's requested features while dependency closure is enforced by the pipeline itself, so the orchestration layer can safely be used in a fully automated setting even at current model performance levels. This is consistent with the broader deployment recommendation emerging from RQ2: RevMine's orchestration layer is most robust when combined with deterministic safeguards that compensate for known LLM weaknesses.

10.5 External Evaluation

To complement the technical evaluations reported in Sections 10.4.1–10.4.3, we solicited formative feedback on RevMine's perceived usefulness from researchers and practitioners outside

the development team. We designed a 15-item online questionnaire covering demographics, evaluation context, and perceived utility along five Likert-scale dimensions, complemented by three free-text items and one recommendation question. Invitations were distributed through the authors' professional network and open practitioner communities. Participation was voluntary and uncompensated.

We received $n = 11$ complete responses. The sample spans six distinct roles (one researcher, one PhD student, three MSc students, four software engineers, one founder/fractional CTO, and one CEO), with experience ranging from less than one year to seven years and GitHub/GitLab usage ranging from *occasionally* to *daily*. Self-reported familiarity with software process analysis covered the full spectrum: three respondents declared themselves *familiar*, five *somewhat familiar*, and three *not familiar*. Three respondents evaluated RevMine hands-on with a real repository; three used the tool via a 30-minute video plus interface exploration; four watched the demonstration video only; one respondent did not specify the evaluation method. Given the small sample size, we report raw distributions and treat all results as formative feedback rather than hypothesis-confirming evidence.

10.5.1 Results

Seven of eleven respondents rated RevMine's overall usefulness $5/5$ *and the remaining four rated it* $4/5$ *(mean* 4.64*), with ease of expressing intent through the natural-language interface averaging* $4.27/5$ *and trust in data fidelity* $4.45/5$ *— externally corroborating the technical-correctness results of Section 10.4.1.* Time to functional understanding of the tool was under 15 minutes for seven of the eleven respondents (three under five minutes, four between five and fifteen), suggesting that the natural-language interface and the unified dashboard effectively reduce the learning curve relative to manual scripting. When asked how RevMine compares to their current data-collection workflow, eight of the eleven respondents answered that it would *"save significant time and effort"* and the remaining three reported that they did not currently collect this type of data, indicating that no respondent saw RevMine as slower or comparable to their existing approach. Table 10.10 summarizes the Likert ratings.

Table 10.10 External-evaluation Likert ratings (n = 11; 1–5 scale)

Dimension	Mean	Median	1	2	3	4	5
Overall usefulness	4.64	5	0	0	0	4	7
Ease of NL interface	4.27	4	0	0	2	4	5
Trust in data fidelity	4.45	5	0	0	1	4	6
Likelihood of future use	4.18	4	0	0	2	5	4
Metric coverage	4.00[†]	4	0	0	2	6	2

[†] One respondent left this item blank (n = 10).

Table 10.11 Recommendation to a colleague (n = 11)

Response	Count
Definitely yes	8
Probably yes	1
Probably not	1
Unsure	1
Definitely no	0

Adoption intent is strong: future-use likelihood averaged 4.18/5*, nine of eleven respondents would recommend RevMine to a colleague (eight* "Definitely yes"*, one* "Probably yes"*), and seven of the eleven reported* no significant barrier to adoption. The two non-recommending responses came from the CEO respondent, who was *unsure* and cited setup complexity, and an MSc student who gave *"Probably not"* after requesting improved LLM parsing of complex requests. The most consistently flagged barrier across the full sample was privacy: three respondents cited concerns about sending repository data to an external LLM, with two of them also flagging LLM parsing accuracy. This privacy-related barrier is directly addressed by RevMine's local-inference Ollama backend (Section 10.4.2), which keeps requests on the user's machine and is the deployment mode recommended in such settings. The remaining barriers (setup complexity, missing metrics) were each cited by at most two respondents and concern scope or operational deployment rather than defects in what is already provided.

The most-valued aspects of RevMine cluster around the built-in visualization and analysis dashboard (cited by all eleven respondents), the natural-language interface (eight), unified GitHub and GitLab support (seven), literature-grounded predefined metrics (seven), and

automatic dependency inference between features and raw metrics (six). No-code accessibility for non-programmer researchers was cited by five respondents, and raw JSON persistence by four — the latter described by one respondent as "enabling reprocessing without re-calling the API", which directly reflects the reproducibility principle described in Section 10.2.

All requested improvements are additive rather than corrective: the most frequent additions are support for additional platforms such as Bitbucket and Azure DevOps (six respondents), CI/CD pipeline data integration (five), and more metrics and computed features (four), with no respondent identifying a defect in the existing functionality. The remaining requests, in descending frequency, were better documentation and usage examples (three), multi-user collaboration features (three), more accurate LLM parsing of complex requests (three), and scheduled or recurring collection (one).

Two respondents volunteered novel use cases that extend beyond the empirical software engineering research context, pointing toward practitioner-oriented applications consistent with RevMine's design but outside its initial intended scope. One suggested running RevMine against legacy codebases to produce technical-health reports ("rework hotspots, review bottlenecks, stale branches") for engineering managers justifying modernisation budgets to non-technical stakeholders; the other requested multi-branch concurrent collection for comparative analysis.

Taken together, the external evaluation indicates that RevMine is perceived as useful, easy to use through its natural-language interface, and trustworthy. Nine of the eleven respondents would recommend it to a colleague. The principal friction points raised were additive rather than corrective, and the most consistently flagged barrier — privacy when sending repository data to an external LLM — is already addressed by RevMine's local Ollama backend. The most frequent single additive request — *CI/CD pipeline data integration* — was raised by five of the eleven respondents and directly motivates the scope extension described in the next section.

10.6 From Code Review to the Full DevOps Pipeline: Kanban and CI/CD Support

The evaluations reported in Sections 10.4.1–10.4.3 establish RevMine as a platform for code review data collection and analysis. However, the prior empirical studies show that code review effort cannot be interpreted in isolation. It is shaped by upstream planning conditions and may be reflected in downstream CI/CD delivery outcomes. Accordingly, we extended RevMine beyond its initial code review scope with preliminary support for two additional DevOps stages: Kanban-style workflow management and CI/CD pipeline execution.

This extension should be interpreted as an initial step toward full end-to-end DevOps analytics. At the current stage, the Kanban and CI/CD modules provide basic collection, normalization, and predefined analyses, but they have not yet received the same level of systematic validation as the core code review collection and orchestration components evaluated in Sections 10.4.1–10.4.3. Their purpose is therefore to make the upstream–review–downstream analytical chain technically expressible inside RevMine, while leaving deeper empirical validation and broader platform coverage as future work.

10.6.1 Kanban Support

The Kanban extension collects issue and board-oriented data from GitHub and GitLab and normalizes them into a platform-independent representation of workflow items. These records capture issue metadata, workflow status, assignee information, labels, and timestamps that approximate when a work item entered and left the tracked workflow. Table 10.12 summarizes the Kanban fields currently collected or computed by RevMine.

On top of these fields, RevMine provides a first set of predefined Kanban analyses. These analyses focus on flow efficiency, throughput, work in progress, blocked work, column-level duration, and workload distribution. Table 10.13 summarizes the currently supported Kanban analyses.

Table 10.12 Kanban metrics currently collected or computed by RevMine

Field	Description
issue_id	Unique identifier of the issue or board item.
title	Title of the issue or workflow item.
status	Current state of the item, such as open, closed, opened, or merged.
column	Current Kanban column, such as To Do, In Progress, or Done, derived from status or custom fields.
created_at	Timestamp indicating when the item was created.
closed_at	Timestamp indicating when the item was closed or resolved.
assignee	Assigned user or users, represented as a comma-separated list.
author	User who created the item.
labels	Labels or tags attached to the item, represented as a comma-separated list.
in_progress_at	Timestamp indicating when the item entered an in-progress state, when available.
done_at	Timestamp indicating when the item was marked as done or closed.
entered_at	Timestamp indicating when the item entered the tracked workflow, approximated by the creation timestamp when detailed transition data are unavailable.
left_at	Timestamp indicating when the item left the tracked workflow, approximated by the closure timestamp when detailed transition data are unavailable.
date	Date field used for temporal aggregation.
duration_h	Computed duration, in hours, between closed_at and created_at.

Table 10.13 Kanban analyses currently supported by RevMine

Analysis code	Computation	Justification
kanban_lead_time	Computes the distribution of the duration between item creation and closure.	Helps identify how long work items remain in the workflow and supports the detection of slow or delayed tasks.
kanban_cycle_time	Computes the time between entering the in-progress state and being marked done.	Captures the active execution time of work items, excluding part of the waiting time before work starts.
kanban_throughput	Counts completed items over time.	Measures the delivery capacity of the team and helps identify variations in productivity across periods.
kanban_wip	Estimates the number of active items in the workflow over time.	Supports the analysis of workload accumulation and helps reveal periods where too many tasks are handled simultaneously.
kanban_cfd	Computes cumulative item counts by period and workflow column.	Provides a flow-oriented view of how work items move across the process and helps reveal bottlenecks between stages.
kanban_column_time	Computes the median duration spent in each Kanban column.	Identifies workflow stages where items tend to remain longer and where process delays may occur.
kanban_blocked_ratio	Estimates the proportion of tracked duration associated with blocked-labelled items.	Measures the extent to which blocked work affects the workflow and helps characterize waiting or dependency-related issues.
kanban_assignee_load	Counts active, non-closed items grouped by assignee.	Supports workload analysis by identifying contributors with high numbers of ongoing tasks.

10.6.2 CI/CD Support

The CI/CD extension collects workflow runs, pipelines, and jobs from GitHub Actions and GitLab CI/CD. The collected fields describe the run or job identifier, execution status, branch, commit SHA, actor, runner, and timing information. These data make it possible to derive

basic delivery indicators such as success rate, build duration, queue time, deployment frequency, and job-level failure profiles. Table 10.14 summarizes the CI/CD fields currently collected or computed by RevMine.

Table 10.14 CI/CD metrics currently collected or computed by RevMine

Field	Description
run_id	Unique identifier of the pipeline or workflow run.
job_id	Unique identifier of the job within a run.
job_name	Name of the job or stage.
workflow_name	Name of the workflow or pipeline.
conclusion	Final result of the run or job, such as success, failure, error, or skipped.
status	Current execution status, such as completed, in progress, or queued.
branch	Git branch associated with the run.
sha	Commit SHA associated with the run.
actor	User who triggered the workflow or pipeline.
runner_name	Runner that executed the job.
created_at	Timestamp indicating when the run or job was created or queued.
started_at	Timestamp indicating when execution started.
completed_at	Timestamp indicating when execution completed.
duration_s	Computed duration, in seconds, between completed_at and started_at.

The predefined CI/CD analyses focus on delivery reliability, execution time, job-level failure patterns, deployment frequency, queueing delay, runner usage, and flaky jobs. Table 10.15 summarizes the CI/CD analyses currently supported by RevMine.

Table 10.15 CI/CD analyses currently supported by RevMine

Analysis code	Computation	Justification
cicd_success_rate	Computes the percentage of successful runs over all CI/CD runs by period.	Provides a high-level indicator of delivery reliability and helps detect periods of unstable integration.
cicd_build_duration	Computes build duration summaries, including median and high-percentile duration values.	Captures the time cost of CI/CD execution and helps identify slow builds or performance regressions.
cicd_failure_rate_by_job	Computes the failure rate for each job or stage.	Helps identify pipeline components that fail frequently and may require maintenance or stabilization.
cicd_mttr	Computes the average time between a failed run and the next successful run.	Measures recovery speed after CI/CD failures and supports analysis of operational responsiveness.
cicd_deploy_frequency	Counts successful deployment-related runs over time.	Provides a delivery-frequency indicator and helps characterize the rhythm of deployment activity.
cicd_queue_time	Computes the waiting time between run creation and execution start.	Captures infrastructure or scheduling delays before execution begins.
cicd_runner_utilization	Aggregates execution duration by runner and time of day.	Supports analysis of runner workload and helps identify uneven use of CI/CD execution resources.
cicd_flaky_jobs	Identifies jobs that produce different conclusions for the same commit SHA.	Helps detect unstable jobs whose outcomes may vary independently of code changes.

Overall, the current DevOps extension adds 15 Kanban fields, including one computed duration field, and 14 CI/CD fields, including one computed duration field. It also adds eight predefined Kanban analyses and eight predefined CI/CD analyses. These analyses use the same analysis interface as the code review analyses: users can request predefined metrics, apply filters, override visualization choices, and aggregate results by day, week, month, quarter, or year.

At this stage, the extension provides basic but useful DevOps-wide visibility. It allows RevMine to connect code review analytics with upstream workflow and downstream delivery data, but it remains an early extension of the platform. In particular, Kanban support is affected by platform heterogeneity: GitLab exposes issue and board workflows more uniformly, whereas GitHub project and board constructs are more variable and only partially covered. Similarly, CI/CD support currently focuses on commonly available workflow and job-level fields rather than deep observability data. A systematic validation of the Kanban and CI/CD modules, using the same level of rigor as the code review evaluation, is therefore left as future work.

10.7 Discussion

RevMine illustrates both the promise and the current limits of using LLMs as an interface for empirical software engineering tools. In the current version, the LLM is not used to freely generate arbitrary analyses. Instead, it helps users explore and activate what already exists in the platform: available data collectors, predefined metrics, supported filters, and implemented analysis routines. This design choice is deliberate. It makes the system more controllable, easier to validate, and less exposed to unverifiable code generation errors. The evaluation therefore measures a bounded form of LLM assistance: whether a natural-language request can be translated into a valid and executable data-collection or analysis plan within the current capabilities of RevMine.

This bounded design also reveals a broader tension for LLM-assisted research tools. On one hand, restricting the LLM to existing functions improves reliability and reproducibility. On the other hand, it limits the user's analytical freedom: if a metric, transformation, or visualization is not

already implemented, the LLM can only approximate the request or reject it. A natural next step is therefore to move from LLM-assisted selection of existing capabilities to controlled generation of missing analysis code. For example, future versions of RevMine could allow the LLM to generate new Python or SQL analysis modules when no predefined metric matches the user's request, while validating the generated code against the available schema, execution sandbox, and reproducibility constraints. Such a direction would turn RevMine from a guided analytics interface into a more flexible research assistant, where users can explore new hypotheses without waiting for every metric to be manually added to the catalogue.

The results also suggest that LLM orchestration should not be evaluated as a single monolithic capability. A model may be strong at producing valid JSON, reasonable metric names, or partially correct extraction plans, but still fail to include all dependencies required for execution. This distinction is important for tool builders. It shows that LLM reliability depends not only on prompt quality or model choice, but also on the surrounding architecture: schema validation, deterministic dependency completion, fallback rules, manual inspection, and execution guards are necessary components of a usable LLM-assisted mining platform. In this sense, RevMine should be understood as a hybrid system where the LLM supports interaction, but correctness must be enforced by explicit software mechanisms.

Another important point concerns the scope of DevOps analytics. The initial focus of RevMine is code review mining, but the extension toward Kanban and CI/CD data shows that review activity is only one part of a larger process chain. Upstream planning affects what reaches review, while downstream CI/CD outcomes provide signals about delivery reliability after review decisions are made. The current Kanban and CI/CD support remains preliminary and less validated than the code review core, but it makes the end-to-end analytical chain technically possible inside a single tool. This is important because many meaningful questions in DevOps cannot be answered from MR data alone.

Finally, RevMine highlights a trade-off between breadth and depth. A broad tool that covers GitHub, GitLab, peer review, Kanban, CI/CD, dashboards, and LLM orchestration can support

many research and industrial scenarios, but each extension introduces new platform differences and validation needs. The value of RevMine therefore lies not in replacing specialized analytics or observability tools, but in providing a reproducible bridge between research questions and multi-source DevOps data.

10.8 Implications for Research and Practice

10.8.1 Implications for practitioners and industrial partners

For practitioners, RevMine provides a reusable entry point for DevOps process analytics. Instead of writing new extraction scripts for each question, teams can use natural-language requests or manual configuration to collect PR/MR, Kanban, and CI/CD data through a common interface. This can support practical activities such as monitoring review delays, identifying large or risky changes, observing workload distribution, tracking blocked work, and relating review behavior to pipeline outcomes.

The tool is especially useful for organizations that want analytical visibility without requiring every manager or analyst to understand the details of GitHub or GitLab APIs. However, the current design should be used as decision support rather than as an automated decision-maker. Because LLM outputs may be incomplete or partially incorrect, generated plans should remain inspectable, reproducible, and validated before use in high-impact industrial decisions.

10.8.2 Implications for researchers

For researchers, RevMine reduces the engineering overhead of empirical studies based on PR/MR data. It provides a shared metric catalogue, a common collection pipeline for GitHub and GitLab, and reusable analysis outputs. This can make replication and cross-project comparison easier, especially when studies require multiple types of traces such as comments, commits, files, reviews, issues, and pipelines.

The tool also provides a concrete architecture for evaluating LLM-assisted research infrastructure. Rather than assessing only final answers, RevMine separates natural-language interpretation, collection planning, dependency resolution, data extraction, and metric computation. This separation makes it possible to evaluate where failures occur and to improve each layer independently. Future work on code-generating LLM agents for empirical software engineering can build on this architecture by adding controlled generation of new metrics and analyses while preserving traceability, sandboxing, and reproducibility.

10.9 Threats to Validity

Internal validity. The 500-scenario benchmark used in RQ2 and RQ3 was manually constructed and may reflect the authors' assumptions about typical user requests. To reduce this risk, three authors contributed to its construction, followed by cross-verification by the first author. The dependency-completeness evaluation relies on a ground-truth dependency map extracted from the current RevMine implementation; therefore, it must be updated as the metric catalogue evolves. For RQ1, residual mismatches were manually inspected, but rare failure modes may remain undetected.

External validity. The 20 repositories used in RQ1 are open-source and may not fully represent industrial projects. The external evaluation sample is small ($n = 11$), so its results should be interpreted as formative rather than representative. The evaluated LLMs also reflect a specific point in time, and their performance may change with model updates or provider availability. RevMine currently supports GitHub and GitLab only; other platforms, such as Gerrit or Bitbucket, require separate validation. Finally, the quantitative evaluation focuses on the code review core. The Kanban and CI/CD extensions are preliminary and have not yet been validated at the same scale.

Construct validity. The RQ2 evaluation dimensions match RevMine's JSON schema and system failure modes, but they do not cover all aspects of LLM output quality. For example, they do not fully assess whether generated filters are semantically appropriate or whether ambiguous user

intent is correctly inferred. The FCR/FAR/CPC metrics used in RQ1 exclude non-deterministic GitHub fields such as `mergeable` and `mergeable_state`, which avoids unfairly penalizing the tool for platform-dependent behavior. The external evaluation relies on self-reported usefulness ratings, which may differ from real task performance.

Conclusion validity. For RQ1, mutable platform fields may change between collection snapshots, although this was analyzed through the mutable-FAR analysis. For RQ2 and RQ3, scoring is automated and deterministic, reducing measurement error. However, the 500-scenario benchmark may be insufficient to detect small differences between similar models. Therefore, model comparisons should be interpreted as indicative rather than definitive. The small external-evaluation sample also limits statistical power; accordingly, we report descriptive results rather than inferential tests.

10.10 Conclusion

This chapter presented RevMine, an LLM-assisted tool for reproducible DevOps-pipeline mining across GitHub and GitLab. RevMine unifies data extraction, metric computation, and natural-language orchestration into a single workflow that covers Kanban-style planning, peer review, and CI/CD delivery, making the kinds of analyses developed throughout this thesis accessible to other researchers and industrial partners as reproducible, cross-stage workflows rather than manually stitched scripts.

The evaluation reported in this chapter supports three main claims. First, RevMine's Data Collection Engine achieves parity with hand-written mining scripts across 20 open-source repositories, with macro-averaged field completeness of 100% and field accuracy of 99.85% on immutable fields, and with residual deviations traceable to platform-level phenomena that would equally affect any asynchronous collector. Second, RevMine's LLM orchestration layer produces schema-compliant JSON in over 92% of cases across nine evaluated models, with intent classification close to ceiling performance and metric recall consistently above 0.73 for the strongest models; strict end-to-end correctness remains limited by the joint satisfaction

of multiple schema fields, and is best managed through the combination of LLM parsing and deterministic post-processing such as automatic dependency closure. Third, external evaluation with eleven respondents from research and industry roles indicates that RevMine is perceived as useful, easy to use, and trustworthy, with friction points concentrated on additive requests (additional platforms, additional metrics) rather than corrective criticisms.

Taken together with the preceding chapters, RevMine closes the thesis's process-oriented view of peer review effort by providing the tooling support needed to transfer the developed analytical methods into reproducible practice, and by extending the unit of analysis from the MR to the full upstream–review–downstream chain. Concretely, the Kanban-stage integration makes the task-estimation and waiting-state analyses of Chapter 8 reproducible on new projects; the CI/CD-stage integration does the same for the self-approval and delivery-reliability analyses of Chapter 9; and the cross-stage join primitives described in Section 10.6 make end-to-end analyses of the full DevOps pipeline expressible as a single natural-language request. The residual limitations identified in this chapter — the pagination cap on very large PRs in RQ1, the strict-FDSR gap in RQ3, the additive feature requests surfaced in the external evaluation, and the need for a systematic FCR/FAR/CPC-style evaluation of the Kanban and CI/CD endpoints — constitute concrete directions for future iterations of the tool.

Acknowledgement: We thank the eleven respondents to the external evaluation questionnaire for their time and feedback, and our industrial partners Kaloom and TELUS for ongoing discussions that shaped RevMine's design priorities.

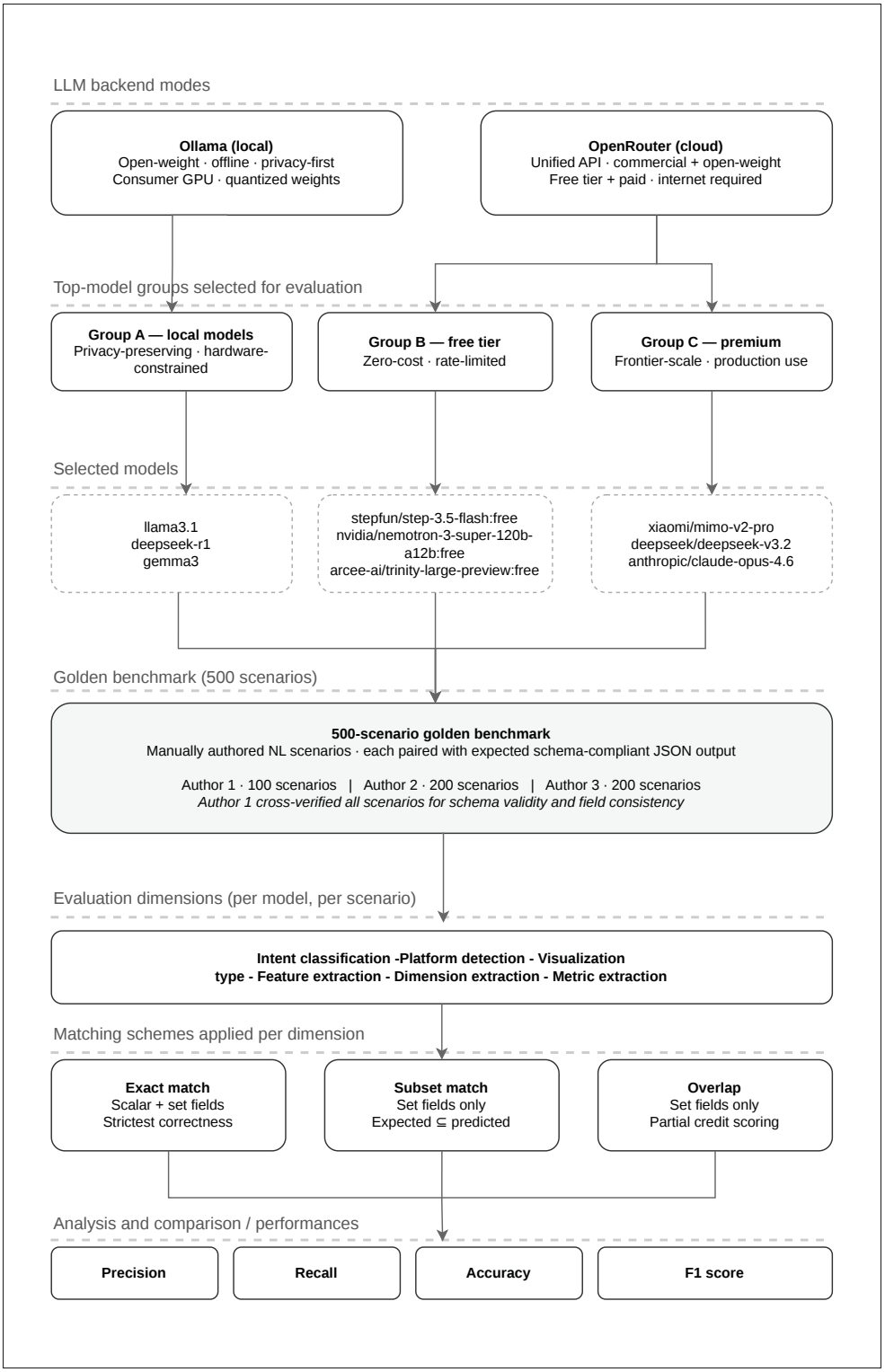


Figure 10.3 Overview of the RevMine LLM evaluation framework

CONCLUSION AND RECOMMENDATIONS

11.1 Global Discussion

Taken together, the seven contribution chapters do more than accumulate evidence on different aspects of peer review. They suggest a particular way of understanding what peer review effort *is* as a research object, and this framing carries consequences that go beyond any individual chapter. This section discusses what the thesis collectively implies about peer review effort, which tensions it exposes, and which limits bound its interpretation.

Peer review effort is a construct, not a metric. Prior work has tended to treat peer review effort as whatever can be measured from a given trace, most often time to complete peer review, and to argue about it as though all these measurements referred to the same underlying quantity. The combined evidence of this thesis suggests instead that peer review effort is a *family* of related but distinct signals: elapsed time, amount of post-submission change, engagement patterns shaped by artifact type, review-interaction intensity, and indirectly, upstream estimation quality and downstream delivery outcomes. Each signal captures a partial view of a shared phenomenon, and different signals correlate differently depending on context. This has a practical consequence for the research community: empirical results that appear contradictory across studies may reflect not contradictory findings but different slices of the same construct. The thesis does not claim that any one of its indicators is the "true" measure of review effort; it argues, by demonstration, that the construct must be approached with multiple co-reported indicators rather than with a single proxy.

"Optimizing" review effort is not synonymous with reducing it. A tension emerges when the chapters are read together. Chapter 7 and Chapter 10.10's downstream analysis identify contexts where review effort can be safely reduced: deviations inflate effort measures without reflecting substantive work, and a substantial share of peer-reviewed MRs are later labelled

as self-approvable without visible downstream failure. Chapter 6, in contrast, shows that configuration-dominant MRs already receive *less* review attention than they arguably deserve, and that bug presence amplifies this asymmetry. Chapter 8 adds that part of the downstream review burden is inherited from upstream underestimation, particularly of small tasks, rather than from review practices themselves. The implication is that optimizing peer review effort cannot be reduced to a single directional goal. It requires *redistributing* effort: relaxing review intensity on MRs where it is empirically unnecessary, while protecting or increasing it where current practice under-invests. A uniform push toward lighter review, driven only by the deviation or self-approval findings, would miss this asymmetry; a uniform push toward heavier review would miss the efficiency findings. The thesis positions optimization as a *calibration* problem rather than a minimization one.

Single-indicator monitoring is insufficient for this calibration. The multi-dimensional framing also has methodological consequences for how teams and researchers monitor review. If time is the only indicator, MRs with low change volume but long elapsed time (e.g., reviewer-availability delays on straightforward MRs) appear indistinguishable from MRs that genuinely triggered heavy rework. If change is the only indicator, AI-assisted revisions or mechanical squashing behaviours (Chapter 5) can be mistaken for substantive review-induced work. If MR type is not controlled for, reviews of configuration and documentation MRs are averaged with development reviews, obscuring both their specific risks and their specific opportunities for effort savings (Chapters 6 and 7). The tooling contribution (Chapter 10) directly operationalizes the opposite stance: RevMine is designed to collect and expose the signals introduced across the thesis in a compatible way, so that multi-indicator monitoring becomes a default rather than an exception. Moreover, because RevMine integrates Kanban-style planning data and CI/CD delivery data alongside peer review data, the joint reporting recommended here is not limited to review-internal dimensions; it can be extended across the full upstream–review–downstream chain, with cross-stage joins (issue → MR → pipeline) exposed as first-class analytical primitives.

Independently of whether RevMine itself is adopted, the conceptual argument applies: peer review effort should be reported jointly along time, change, type, and deviation dimensions, and ideally also against the planning conditions that precede it and the delivery outcomes that follow from it. This joint reporting is a non-trivial precondition for replication and meta-analysis across studies.

Peer review is most informative when analyzed as a pipeline stage, not a pipeline. The preceding chapters also show that several of the most actionable findings only become visible once peer review is situated inside the DevOps pipeline rather than studied as a standalone activity. The observability findings (Chapter 4) already position review traces as part of a broader process signal covering environmental disruptions, technological transitions, and bot–human interaction. Chapter 8 shows that part of the review burden observed downstream is inherited upstream, from Kanban-style task estimation and waiting-state exposure. Chapter 9 shows that review decisions, notably self-approval, cannot be evaluated for appropriateness without linking them to CI/CD outcomes such as pipeline success and post-merge failure time. Read jointly, these chapters reframe peer review as a *stage* embedded in a three-node Kanban → review → CI/CD chain, where the information needed to judge a review decision often lives at an adjacent stage. RevMine (Chapter 10) is the direct methodological consequence of this framing: its Kanban-, review-, and CI/CD-collection subsystems are not parallel silos but an integrated platform in which cross-stage joins (issue ID → closing-MR; MR merge SHA → pipeline) are part of the collection plan itself. The broader methodological implication for the community is that empirical studies of review effort should, wherever possible, co-report the upstream planning signals and downstream delivery signals associated with the reviews they study. The thesis demonstrates, through the concrete analyses and through the tooling, that this is both feasible and analytically productive.

The approach is bounded by what traces can show. Despite this breadth, the thesis observes only what MR traces record. Review-related work that occurs off-trace — direct messages, pair-review sessions, standups, informal escalations, or AI assistance whose intermediate steps are not logged — remains invisible. The trace-based view therefore underestimates total review effort in settings where such channels are heavily used, and the magnitude of this underestimation is itself not uniform across organizations. A related limit concerns interpretation: even well-defined trace signals, such as self-approval or workflow deviations, depend on local review and merge conventions that differ across organizations, which bounds the transferability of any specific numerical threshold derived from the thesis's datasets. The contributions are therefore best read as *methods and constructs* that transfer across contexts, rather than as *parameter values* that transfer directly.

Context dependence and the temporal drift of the construct. The thesis combines two industrial contexts (Kaloom, TELUS) with open-source GitLab projects, and the patterns are not identical across them. The amount-of-change metric affects up to 73% of open-source MRs but only 18% of industrial ones; deviation distributions differ across MR types; self-approval rates and their downstream consequences vary sharply by project. These divergences are themselves informative — they confirm that review effort cannot be studied as a universal phenomenon detached from organizational setting — but they also imply that practitioners and researchers using the thesis's constructs need to characterize their own context before applying its conclusions. A second contextual limit concerns time: the AI-assisted development regime that is already reshaping code authoring (Chapter 5's discussion) is only beginning to appear in the datasets used. The thesis should therefore be read as establishing the construct, the measurement apparatus, and the baseline patterns *before* generative assistance became dominant; the same apparatus can be re-applied as that regime matures, but the parameter values it yields are likely to evolve.

Convergent evidence for a strong combined contribution. Despite these qualifications, the combined contribution of the thesis is robust for a specific reason: it does not rest on a single analytical setup. Seven chapters, built on distinct datasets, distinct analytical methods, and distinct dimensions of the review process, all converge on the same core message: peer review effort is multi-dimensional, context-dependent, shaped by upstream and downstream conditions, and amenable to principled calibration rather than blind reduction. Each individual chapter, in isolation, could be reasonably contested on dataset scope, measurement choice, or context specificity; together, they would have to be contested jointly, which is a much harder empirical claim to sustain. The methodological artifacts produced along the way — the change-based effort measure, the MR-type classification, the deviation taxonomy and detector, the reference lead time framework, the self-approval recommender, and the RevMine tool — are reusable independently of the specific findings reported here, and they make the thesis's framing operationalizable in settings that go beyond its own empirical scope. On this basis, the thesis advances a view of peer review effort that is at once more structured than the prevailing time-centric framing, more careful in how it aggregates across MR types and workflows, and more connected to the surrounding software process, while remaining explicit about the boundaries within which these advances hold.

11.2 Global Threats to Validity

Several threats apply to the thesis as a whole and complement the chapter-specific threats discussed in each chapter.

Generalizability across organizations and platforms. Most of the empirical material comes from two industrial partners, TELUS and Kaloom, supplemented by a limited set of open-source GitLab projects. Organizations with different sizes, domains, development cultures, or review policies may exhibit different patterns. Similarly, the thesis focuses on the GitLab MR mechanism; although RevMine unifies GitHub and GitLab, most empirical analyses rely on

GitLab data. Other review platforms, such as Gerrit or GitHub Pull Requests, may differ in ways that affect how the reported findings transfer. Where possible, the chapters include replication on open-source projects to broaden the empirical basis, but further industrial and open-source replications remain necessary.

Observational nature of the analyses. All empirical analyses in the thesis are observational. Although they use large datasets, principled metric definitions, and statistical tests, they do not control for every potential confounder, and associations should not be interpreted as causal effects. This concerns the relations between MR characteristics and review effort, between deviations and predictive-model behavior, between upstream planning conditions and downstream review burden, and between review decisions and CI/CD outcomes.

Heuristic classifications and manual annotations. Several key constructs rely on heuristics or manual annotations: MR type classification (configuration, development, documentation), deviation categorization, bug identification, and self-approvable MR labeling. These choices are validated with industry experts or against prior literature, but heuristics may not transfer perfectly to other repositories, and manual annotation may introduce subjectivity. The thesis mitigates this by documenting all criteria, sharing replication packages, and relying on automated, deterministic pipelines where possible.

Temporal stability of the findings. Industrial DevOps contexts evolve continuously, and some of the empirical material spans specific periods (e.g., the COVID-19 disruption, the OpenShift migration). Practices, tooling, and AI-assisted development continue to change, which may alter the relationships observed between planning, review effort, and delivery outcomes over time. This concern is explicitly discussed with respect to tool-assisted development but applies more broadly to the whole set of findings.

11.3 Global Conclusion

This thesis studied peer review effort as a software process problem in DevOps environments and asked how it can be better characterized, analyzed, and optimized. Starting from the observation that peer review effort has traditionally been approximated through temporal measures alone, the thesis built an expanded, process-oriented view in seven steps. It established peer review traces as a rich empirical basis for studying software process behavior, introduced a time-independent change-based indicator of review effort, characterized how this effort varies across MR types, identified workflow deviations as an important bias source, linked peer review effort to upstream Kanban-style planning conditions, related peer review decisions to downstream CI/CD delivery reliability, and delivered RevMine, a reusable tool for reproducible DevOps-pipeline analytics that integrates Kanban planning, peer review, and CI/CD delivery data across Git-based platforms into a single cross-stage workflow.

Practical impact. For our industrial partners, the thesis offers concrete recommendations for monitoring peer review effort beyond elapsed time, applying deviation-aware analytics, aligning review practices with MR type and bug presence, calibrating estimation practices to reduce downstream rework, and using creation-time information to support risk-aware review decisions. RevMine packages these analytical capabilities into a reusable tool that lowers the adoption barrier in practice, and its DevOps-pipeline scope (Kanban $\rightarrow$ peer review $\rightarrow$ CI/CD) means the same tool can support upstream estimation dashboards, effort-oriented review monitoring, and post-merge reliability analyses within a single analytical workflow.

Future directions. Several directions emerge naturally from the thesis. First, broader replication across additional industrial and open-source contexts and across review platforms beyond GitLab would strengthen external validity. Second, moving from observational analysis to designed interventions, such as controlled rollouts of deviation-aware dashboards or self-approval recommendations, would let researchers test the causal impact of the proposed methods. Third,

the rise of AI-assisted development invites re-examination of effort indicators, as tool-assisted coding may change the relationship between estimated size, implementation effort, and review burden. Fourth, now that RevMine integrates Kanban planning and CI/CD delivery alongside peer review, a natural follow-up is to systematically evaluate the Kanban and CI/CD collection behaviours with the same FCR/FAR/CPC correctness protocol used for PRs/MRs, and to apply the tool's cross-stage joins in full-pipeline digital-twin scenarios where planning, review, and delivery signals are monitored jointly in real time. Finally, extending RevMine with additional review platforms (Bitbucket, Azure DevOps, Gerrit) and connecting it to industrial dashboards would enable continuous process-intelligence use cases that build directly on the analyses developed in this thesis.

Overall, this thesis advances the understanding of peer review effort as a measurable and optimizable software process phenomenon. By linking it to upstream planning, artifact characteristics, workflow regularity, and downstream delivery, and by providing tooling support to transfer the methods into practice, the thesis lays an empirical and methodological foundation for future efforts to optimize peer review in DevOps environments.

BIBLIOGRAPHY

Adler, R. F. & Benbunan-Fich, R. (2012). Juggling on a high wire: Multitasking effects on performance. *International Journal of Human-Computer Studies*, 70(2), 156–168. doi: 10.1016/j.ijhcs.2011.10.003.

Agarwal, A. & Gupta, N. (2021). Comparison of outlier detection techniques for structured data. *arXiv preprint arXiv:2106.08779*.

Agrawal, A. & Menzies, T. (2018). Is" Better Data" Better Than" Better Data Miners"? *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pp. 1050–1061.

Agrawal, A. & Menzies, T. (2019). Is AI different for SE? *NC State Univ., Aug*, 26.

Aguilera-Martos, I., Luengo, J. & Herrera, F. (2023). Revisiting Histogram Based Outlier Scores: Strengths and Weaknesses. *International Conference on Hybrid Artificial Intelligence Systems*, pp. 39–48.

Aguinis, H., Gottfredson, R. K. & Joo, H. (2013). Best-practice recommendations for defining, identifying, and handling outliers. *Organizational Research Methods*, 16(2), 270–301.

Ahmad, M. O., Markkula, J. & Oivo, M. (2013). Kanban in software development: A systematic literature review. *2013 39th Euromicro conference on software engineering and advanced applications*, pp. 9–16.

Ahsan, S. N. & Wotawa, F. (2010). Impact analysis of SCRs using single and multi-label machine learning classification. *Proceedings of the 2010 ACM-IEEE international symposium on empirical software engineering and measurement*, pp. 1–4.

Aïdasso, H., Bordeleau, F. & Tizghadam, A. (2025). Towards Build Optimization Using Digital Twins. *Proceedings of the 21st International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE)*.

Akar, Ö. & Güngör, O. (2012). Classification of multispectral images using Random Forest algorithm. *Journal of Geodesy and Geoinformation*, 1(2), 105–112.

Akoglu, H. (2018). User's guide to correlation coefficients. *Turkish journal of emergency medicine*, 18(3), 91–93.

Al-Fraihat, D., Sharrab, Y., Al-Ghuwairi, A.-R., Alzabut, H., Beshara, M. & Algarni, A. (2024). Utilizing machine learning algorithms for task allocation in distributed agile software development. *Heliyon*, 10(21), e39926. doi: 10.1016/j.heliyon.2024.e39926.

Albrecht, A. & Gaffney, J. (1983). Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. *IEEE Transactions on Software Engineering*, SE-9(6), 639–648. doi: 10.1109/TSE.1983.235271. Conference Name: IEEE Transactions on Software Engineering.

Ali, A., Khan, N., Abu-Tair, M., Noppen, J., McClean, S. & McChesney, I. (2021). Discriminating features-based cost-sensitive approach for software defect prediction. *Automated Software Engineering*, 28(2), 1–18.

Allahyari, H. & Lavesson, N. (2011). User-oriented assessment of classification model understandability. *11th scandinavian conference on Artificial intelligence*.

AlOmar, E. A. (2025). Deciphering refactoring branch dynamics in modern code review: An empirical study on qt. *Information and Software Technology*, 177, 107596.

AlOmar, E. A., AlRubaye, H., Mkaouer, M. W., Ouni, A. & Kessentini, M. (2021). Refactoring practices in the context of modern code review: An industrial case study at Xerox. *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 348–357.

Alon, U., Sadaka, R., Levy, O. & Yahav, E. (2020). Structural language models of code. *International conference on machine learning*, pp. 245–256.

Alsaadi, B. & Saeedi, K. (2022). Data-driven effort estimation techniques of agile user stories: a systematic literature review. *Artificial Intelligence Review*, 55(7), 5485–5516.

Altman, D. G. & Royston, P. (2006). The cost of dichotomising continuous variables. *Bmj*, 332(7549), 1080.

Alturki, S. & E-amin, F. (2025). Comprehensive Analysis of Software Effort Estimation Techniques: Evolving Trends, Key Challenges, and Prospective Directions. *International Journal of Computer Applications*, 186(68), 42–48.

Amrit, C. (2005). Coordination in software development: the problem of task allocation. *Proceedings of the 2005 workshop on Human and social factors of software engineering*, (HSSE '05), 1–7. doi: 10.1145/1083106.1083107.

Angiulli, F. & Fassetti, F. (2007). Detecting distance-based outliers in streams of data. *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pp. 811–820.

Antoniadis, A., Lambert-Lacroix, S. & Poggi, J.-M. (2021). Random forests for global sensitivity analysis: A selective review. *Reliability Engineering & System Safety*, 206, 107312.

Arar, Ö. F. & Ayan, K. (2015). Software defect prediction using cost-sensitive neural network. *Applied Soft Computing*, 33, 263–277.

Arenas, C., Toma, C., Cormand, B. & Irigoien, I. (2017). Identifying extreme observations, outliers and noise in clinical and genetic data. *Current Bioinformatics*, 12(2), 101–117.

Arifin, H. H., Daengdej, J. & Khanh, N. T. (2017). An Empirical Study of Effort-Size and Effort-Time in Expert-Based Estimations. *2017 8th International Workshop on Empirical Software Engineering in Practice (IWESEP)*, pp. 35–40. doi: 10.1109/IWESEP.2017.21.

Armah, G. K., Luo, G., Qin, K. & Mbandu, A. S. (2016). Applying variant variable regularized logistic regression for modeling software defect predictor. *Lecture Notes on Software Engineering*, 2(4), 107–115.

Aslam, W. & Ijaz, F. (2018). A Quantitative Framework for Task Allocation in Distributed Agile Software Development. *IEEE Access*, 6, 15380–15390. doi: 10.1109/ACCESS.2018.2803685. Conference Name: IEEE Access.

Authors, A. [Anonymized for double-anonymous review]. (2025). On the Impact of Merge Request Deviations on Code Review Practices.

Avula, S. K. & Mondal, S. (2024). MineCPP: Mining Bug Fix Pairs and Their Structures. *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, pp. 552–556.

Aydin, A. & Tarhan, A. (2014). Investigating defect prediction models for iterative software development when phase data is not recorded lessons learned. *2014 9th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, pp. 1–11.

Bacchelli, A. & Bird, C. (2013). Expectations, Outcomes, and Challenges of Modern Code Review. *ICSE*, pp. 712–721. doi: 10.1109/ICSE.2013.6606617.

Bachmann, A. & Bernstein, A. (2010). When process data quality affects the number of bugs: Correlations in software engineering datasets. *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pp. 62–71.

Badampudi, D., Unterkalmsteiner, M. & Britto, R. (2023). Modern code reviews—Survey of literature and practice. *ACM Transactions on Software Engineering and Methodology*, 32(4), 1–61.

Bahel, V., Pillai, S. & Malhotra, M. (2020). A comparative study on various binary classification algorithms and their improved variant for optimal performance. *2020 IEEE Region 10 Symposium (TENSYMP)*, pp. 495–498.

Bal, P. R. & Kumar, S. (2022). A Data Transfer and Relevant Metrics Matching based Approach for Heterogeneous Defect Prediction. *IEEE Transactions on Software Engineering*.

Bansal, A. & Jain, A. (2021). Analysis of Focussed Under-Sampling Techniques with Machine Learning Classifiers. *2021 IEEE/ACIS 19th International Conference on Software Engineering Research, Management and Applications (SERA)*, pp. 91–96.

Barnett, V. & Lewis, T. (1984). Outliers in statistical data. *Wiley Series in Probability and Mathematical Statistics. Applied Probability and Statistics*.

Batini, C., Cappiello, C., Francalanci, C. & Maurino, A. (2009). Methodologies for data quality assessment and improvement. *ACM Computing Surveys (CSUR)*, 41(3), 1–52.

Battina, D. S. (2021). Improving La Redoute's CI/CD Pipeline and DevOps Processes by Applying Machine Learning Techniques. *International Journal of Emerging Technologies and Innovative Research (www. jetir. org| UGC and issn Approved), ISSN*, 2349–5162.

Bavota, G. & Russo, B. (2015). Four eyes are better than two: On the impact of code reviews on software quality. *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 81–90.

Baysal, O., Kononenko, O., Holmes, R. & Godfrey, M. W. (2016). Investigating technical and non-technical factors influencing modern code review. *Empirical Software Engineering*, 21, 932–959.

Beller, M., Bacchelli, A., Zaidman, A. & Juergens, E. (2014). Modern code reviews in open-source projects: Which problems do they fix? *Proceedings of the 11th working conference on mining software repositories*, pp. 202–211.

Bello, R. & Falcon, R. (2017). Rough sets in machine learning: a review. *Thriving Rough Sets*, 87–118.

Beltran, S. & Cardozo, N. (2018). A collaborative IDE for dynamic multi-versioning and variant management. *CIbSE*, pp. 437–450.

Benala, T. R. & Tantati, K. (2022). Efficiency of oversampling methods for enhancing software defect prediction by using imbalanced data. *Innovations in Systems and Software Engineering*, 1–17.

Bennin, K. E., Keung, J., Phannachitta, P., Monden, A. & Mensah, S. (2017). Mahakil: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction. *IEEE Transactions on Software Engineering*, 44(6), 534–550.

Benzekki, K., El Fergougui, A. & Elbelrhiti Elalaoui, A. (2016). Software-defined networking (SDN): a survey. *Security and communication networks*, 9(18), 5803–5833.

Bessghaier, N., Ouni, A., Sayagh, M., Chouchen, M. & Mkaouer, M. W. (2025). Towards understanding code review practices for infrastructure-as-code: An empirical study on OpenStack projects. *Empirical Software Engineering*, 30(3), 106.

Bilgin, Z., Ersoy, M. A., Soykan, E. U., Tomur, E., Çomak, P. & Karaçay, L. (2020). Vulnerability prediction from source code using machine learning. *IEEE Access*, 8, 150672–150684.

Bird, C., Nagappan, N., Murphy, B., Gall, H. & Devanbu, P. (2011). Don't touch my code! Examining the effects of ownership on software quality. *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pp. 4–14.

Bird, C., Carnahan, T. & Greiler, M. (2015). Lessons learned from building and deploying a code review analytics platform. *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pp. 191–201.

Birrell, A. D. (1989). *An introduction to programming with threads*. Digital Systems Research Center.

Biswas, S. & Rajan, H. (2021). Fair preprocessing: towards understanding compositional fairness of data transformers in machine learning pipeline. *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 981–993.

Bosch, N. & Bosch, J. (2020). Software Logs for Machine Learning in a DevOps Environment. *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 29–33.

Bosu, A. & Carver, J. C. (2013). Impact of peer code review on peer impression formation: A survey. *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 133–142.

Bosu, A. & Carver, J. C. (2014). Impact of developer reputation on code review outcomes in oss projects: An empirical investigation. *Proceedings of the 8th ACM/IEEE international symposium on empirical software engineering and measurement*, pp. 1–10.

Bosu, A., Carver, J. C., Hafiz, M., Hilley, P. & Janni, D. (2014). Identifying the characteristics of vulnerable code changes: An empirical study. *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*, pp. 257–268.

Bosu, A., Greiler, M. & Bird, C. (2015). Characteristics of useful code reviews: An empirical study at microsoft. *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pp. 146–156.

Boukerche, A., Zheng, L. & Alfandi, O. (2020). Outlier detection: Methods, models, and classification. *ACM Computing Surveys (CSUR)*, 53(3), 1–37.

Bouraffa, A., Brandt, C., Zaidman, A. & Maalej, W. (2025). Not One to Rule Them All: Mining Meaningful Code Review Orders From GitHub. *Proceedings of the 29th International Conference on Evaluation and Assessment in Software Engineering*, pp. 349–359.

Breiman, L., Friedman, J. H., Olshen, R. A. & Stone, C. J. (2017). *Classification and regression trees*. Routledge.

Brice, J. (2021). Ali Tizghadam | Mobile Magazine. Retrieved on 2025-01-24 from: https://mobile-magazine.com/interviews/ali-tizghadam-principal-technology-architect-telus-0.

Brown, A., Forsgren, N., Humble, J., Kersten, N. & Kim, G. (2016). state of DevOps report. *Puppet+ DORA*.

Brown, J. (2018). Classifiers and their metrics quantified. *Molecular informatics*, 37(1-2), 1700127.

Buser, T. & Peter, N. (2012). Multitasking. *Experimental Economics*, 15(4), 641–655. doi: 10.1007/s10683-012-9318-8.

Cais, Š. & Pícha, P. (2014). Identifying software metrics thresholds for safety critical system. *The Third International Conference on Informatics Engineering and Information Science (ICIEIS2014), The Society of Digital Information and Wireless Communications*, pp. 67–78.

Çakır, H., İncereis, N., Akgün, B. T. & Taştemir, A. S. Y. (2021). Comparison of Sampling Methods Using Machine Learning and Deep Learning Algorithms with an Imbalanced Data Set for the Prevention of Violence Against Physicians. *2021 15th Turkish National Software Engineering Symposium (UYMS)*, pp. 1–7.

Cao, L. (2022). Estimating efforts for various activities in agile software development: An empirical study. *IEEE Access*, 10, 83311–83321.

Capizzi, A., Distefano, S., Araújo, L. J., Mazzara, M., Ahmad, M. & Bobrov, E. (2020). Anomaly detection in devops toolchain. *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment: Second International Workshop, DEVOPS 2019, Château de Villebrumier, France, May 6–8, 2019, Revised Selected Papers 2*, pp. 37–51.

Cassee, N., Vasilescu, B. & Serebrenik, A. (2020). The Silent Helper: The Impact of Continuous Integration on Code Reviews. *SANER*, pp. 423–433. doi: 10.1109/SANER48275.2020.9054818.

Castelluccio, M., Sansone, C., Verdoliva, L. & Poggi, G. (2017). Automatically analyzing groups of crashes for finding correlations. *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pp. 717–726.

Catak, T., Durdu, P. O. & Omurca, S. I. (2024). Enhancing agile effort estimation: An nlp approach for software requirements analysis. *2024 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, pp. 1–8.

Chakkarwar, V., Tamane, S. & Thombre, A. (2023). A Review on BERT and Its Implementation in Various NLP Tasks. *International Conference on Applications of Machine Intelligence and Data Analytics (ICAMIDA 2022)*, pp. 112–121.

Chakravorty, G., Reddy, B. R. & khan, D. A. (2025). Enhancing effort and estimation in scrum-based agile projects with a proposed federated agile framework. *International Journal of Information Technology*, 1–13.

Chandola, V., Banerjee, A. & Kumar, V. (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3), 1–58.

Charoenwet, W., Thongtanunam, P., Pham, V.-T. & Treude, C. (2024). Toward effective secure code reviews: an empirical study of security-related coding weaknesses. *Empirical Software Engineering*, 29(4), 88.

Cheemaa, A. S., Azhar, M., Arif, F., Sohail, M., Iqbal, A. et al. (2025). EGPT-SPE: story point effort estimation using improved GPT-2 by removing inefficient attention heads. *Applied Intelligence*, 55(15), 1–16.

Chen, L., Rigby, P. C. & Nagappan, N. (2022). Understanding why we cannot model how long a code review will take: an industrial case study. *Proceedings of the 30th ACM Joint European software engineering conference and symposium on the foundations of software engineering*, pp. 1314–1319.

Chen, W.-R., Yun, Y.-H., Wen, M., Lu, H.-M., Zhang, Z.-M. & Liang, Y.-Z. (2016). Representative subset selection and outlier detection via isolation forest. *Analytical methods*, 8(39), 7225–7231.

Chen, Z., Kommrusch, S., Tufano, M., Pouchet, L.-N., Poshyvanyk, D. & Monperrus, M. (2019). Sequencer: Sequence-to-sequence learning for end-to-end program repair. *IEEE Transactions on Software Engineering*, 47(9), 1943–1959.

Cheng, Z., Zou, C. & Dong, J. (2019). Outlier detection using isolation forest and local outlier factor. *Proceedings of the conference on research in adaptive and convergent systems*, pp. 161–168.

Chicco, D. & Jurman, G. (2020). The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC genomics*, 21, 1–13.

Chouchen, M. & Ouni, A. (2024). A multi-objective effort-aware approach for early code review prediction and prioritization. *Empirical Software Engineering*, 29(1), 29.

Chouchen, M., Ouni, A., Olongo, J. & Mkaouer, M. W. (2023). Learning to Predict Code Review Completion Time In Modern Code Review. *Empirical Software Engineering*, 28(4), 82.

Ciniselli, M., Cooper, N., Pascarella, L., Poshyvanyk, D., Di Penta, M. & Bavota, G. (2021). An empirical study on the usage of bert models for code completion. *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pp. 108–119.

Cito, J., Dillig, I., Kim, S., Murali, V. & Chandra, S. (2021). Explaining mispredictions of machine learning models using rule induction. *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 716–727.

Coelho, E. & Basu, A. (2012). Effort estimation in agile software development using story points. *International Journal of Applied Information Systems (IJAIS)*, 3(7), 7–10. Publisher: Citeseer.

Cohen, J. (1983). The cost of dichotomization. *Applied psychological measurement*, 7(3), 249–253.

Cohen, W. (1996). Learning Rules that Classify E-Mail in AAAI Spring Symposium on Machine Learning and Information Retrieval, 18-25. Menlo Park, AAAI Press.

Cois, C. A., Yankel, J. & Connell, A. (2014). Modern DevOps: Optimizing software development through effective system interactions. *2014 IEEE international professional communication conference (IPCC)*, pp. 1–7.

Colley, D., Stanier, C. & Asaduzzaman, M. (2018). The Impact of Object-Relational Mapping Frameworks on Relational Query Performance. *2018 International Conference on Computing, Electronics & Communications Engineering (iCCECE)*, pp. 47–52. doi: 10.1109/iCCECOME.2018.8659222.

Corona, E. & Pani, F. E. (2013). A review of lean-kanban approaches in the software development. *WSEAS transactions on information science and applications*, 10(1), 1–13.

Cortes, C. (2004). AUC optimization vs. error rate minimization. Research Report, AT&T Labs. *http://books. nips. cc/papers/files/nips16/NIPS2003_AA40. pdf*.

Costa, J. F. (2003). Reducing the impact of outliers in ore reserves estimation. *Mathematical geology*, 35(3), 323–345.

Cousineau, D. & Chartier, S. (2010). Outliers detection and treatment: a review. *International Journal of Psychological Research*, 3(1), 58–67.

CRAN. [[Accessed 20-Dec-2022]]. (2022a). Ckmeans.1d.dp: Optimal, Fast, and Reproducible Univariate Clustering — cran.r-project.org. Retrieved from: https://cran.r-project.org/web/packages/Ckmeans.1d.dp.

CRAN. [[Accessed 20-Dec-2022]]. (2022b). rpart: Recursive Partitioning and Regression Trees — cran.r-project.org. Retrieved from: https://cran.r-project.org/web/packages/rpart.

CRAN. [[Accessed 20-Dec-2022]]. (2022c). ScottKnottESD: The Scott-Knott Effect Size Difference (ESD) Test — cran.r-project.org. Retrieved from: https://cran.r-project.org/web/packages/ScottKnottESD/.

CRAN. (2026). Package Kendall. Retrieved from: https://cran.r-project.org/web/packages/Kendall/index.html.

Critical Data, M. (2016). *Secondary analysis of electronic health records*. Springer Nature.

Da Costa, D. A., McIntosh, S., Shang, W., Kulesza, U., Coelho, R. & Hassan, A. E. (2016). A framework for evaluating the results of the szz approach for identifying bug-introducing changes. *IEEE Transactions on Software Engineering*, 43(7), 641–657.

Dalla Palma, S., van Asseldonk, C., Catolino, G., Di Nucci, D., Palomba, F. & Tamburri, D. A. (2024). "Through the looking-glass. . . " An empirical study on blob infrastructure blueprints in the Topology and Orchestration Specification for Cloud Applications. *Journal of Software: Evolution and Process*, 36(4), e2533.

Dam, H. K., Tran, T. & Ghose, A. (2018). Explainable software analytics. *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*, pp. 53–56.

Dawson, N. V. & Weiss, R. (2012). Dichotomizing continuous variables in statistical analysis: a practice to avoid.

de Almeida, M. A., Lounis, H. & Melo, W. L. (1998). An investigation on the use of machine learned models for estimating correction costs. *Proceedings of the 20th international conference on software engineering*, pp. 473–476.

De Souza, C. R., Redmiles, D., Cheng, L.-T., Millen, D. & Patterson, J. (2004). How a good software practice thwarts collaboration: the multiple roles of APIs in software development. *Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering*, pp. 221–230.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K. & Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6), 391–407.

Dempster, A. P., Laird, N. M. & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1), 1–22.

Devlin, J. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dima, A. M. & Maassen, M. A. (2018). From Waterfall to Agile software: Development models in the IT sector, 2006 to 2018. Impacts on company management. *Journal of International Studies*, 11(2), 315–325. Retrieved from: https://www.ceeol.com/search/article-detail?id=718102. Publisher: Fundacja Centrum Badań Socjologicznych.

Divya, D. & Babu, S. S. (2016). Methods to detect different types of outliers. *2016 International Conference on Data Mining and Advanced Computing (SAPIENCE)*, pp. 23–28.

Docs. (2026). Merge requests | GitLab — docs.gitlab.com. Retrieved from: https://docs.gitlab.com/ee/user/project/merge_requests/.

Doğan, E. & Tüzün, E. (2022). Towards a taxonomy of code review smells. *Information and Software Technology*, 142, 106737.

Domingos, P. (2026). A general method for making classifiers cost-sensitive. *Artificial Inelligence Group, Instituto Superior Técnico, Lisboa*, 1049–001.

Drosos, G.-P., Sotiropoulos, T., Alexopoulos, G., Mitropoulos, D. & Su, Z. (2024). When your infrastructure is a buggy program: Understanding faults in infrastructure as code ecosystems. *Proceedings of the ACM on Programming Languages*, 8(OOPSLA2), 2490–2520.

Duc Anh, N., Cruzes, D. S., Conradi, R. & Ayala, C. (2011). Empirical validation of human factors in predicting issue lead time in open source projects. *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, pp. 1–10.

Duggan, J., Byrne, J. & Lyons, G. (2004). A task allocation optimizer for software construction. *IEEE Software*, 21(3), 76–82. doi: 10.1109/MS.2004.1293077. Conference Name: IEEE Software.

Dyer, R., Nguyen, H. A., Rajan, H. & Nguyen, T. N. (2013). Boa: A Language and Infrastructure for Analyzing Ultra-Large-Scale Software Repositories. *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, pp. 422–431.

D'Ambros, M., Lanza, M. & Robbes, R. (2012). Evaluating defect prediction approaches: a benchmark and an extensive comparison. *Empirical Software Engineering*, 17, 531–577.

Ebert, F., Castor, F., Novielli, N. & Serebrenik, A. (2021). An exploratory study on confusion in code reviews. *Empirical Software Engineering*, 26(1), 12.

Ekinci, E., Omurca, S. İ. & Acun, N. (2018). A comparative study on machine learning techniques using Titanic dataset. *7th international conference on advanced technologies*, pp. 411–416.

El-Emam, K., Goldenson, D., McCurley, J. & Herbsleb, J. (2001). Modelling the likelihood of software process improvement: An exploratory study. *Empirical Software Engineering*, 6(3), 207–229.

En. (2026). Telus - Wikipedia — en.wikipedia.org. Retrieved from: https://en.wikipedia.org/wiki/Telus.

Escalante, H. J. (2005). A comparison of outlier detection algorithms for machine learning. *Proceedings of the International Conference on Communications in Computing*, pp. 228–237.

Fabriek, M., Brand, M., Brinkkemper, S., Harmsen, F. & Helms, R. (2008). Reasons for Success and Failure in Offshore Software Development Projects. *16th European Conference on Information Systems, ECIS 2008*, 446-457.

Falessi, D., Huang, J., Narayana, L., Thai, J. F. & Turhan, B. (2020). On the need of preserving order of data when validating within-project defect classifiers. *Empirical Software Engineering*, 25(6), 4805–4830.

Fan, Y., Xia, X., Lo, D. & Li, S. (2018). Early prediction of merged code changes to prioritize reviewing tasks. *Empirical Software Engineering*, 23, 3346–3393.

Fan, Y., Xia, X., Da Costa, D. A., Lo, D., Hassan, A. E. & Li, S. (2019). The impact of mislabeled changes by szz on just-in-time defect prediction. *IEEE transactions on software engineering*, 47(8), 1559–1586.

Feamster, N., Rexford, J. & Zegura, E. (2014). The road to SDN: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review*, 44(2), 87–98.

Fenton, N. E., Neil, M. & Square, N. (2005). A critique of software defect prediction models. *SERIES ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING*, 16, 72.

Fister, J. M. (2007). *Correlation analysis of on-page attributes and search engine rankings*. (Master's thesis, University of Cincinnati).

Flach, P. A. (2003). The geometry of ROC space: understanding machine learning metrics through ROC isometrics. *Proceedings of the 20th international conference on machine learning (ICML-03)*, pp. 194–201.

Folleco, A., Khoshgoftaar, T. M., Van Hulse, J. & Bullard, L. (2008). Software quality modeling: The impact of class noise on the random forest classifier. *2008 IEEE congress on evolutionary computation (IEEE world congress on computational intelligence)*, pp. 3853–3859.

Fonseca, J. & Bacao, F. (2026). Geometric SMOTE for Imbalanced Datasets with Nominal and Continuous Features. *Available at SSRN 4183410*.

Forsgren, N., Humble, J. & Kim, G. (2018). *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations*. IT Revolution.

Foundation, O. N. (2012). Software Defined Networking: The New Norm for Networks. Retrieved from: Softwaredefinednetworking:Thenewnormfornetworks,PaloAlto,CA, USA,WhitePaper,Apr.2012.[Online].Available:https://www.opennetworking.org/ images/stories/downloads/white-papers/wp-sdnnewnorm.pdf.

Foundation, O. N. (2023). Open Networking Foundation. Retrieved from: OpenNetworkingFoundation(ONF).[Online].Available:https: //www.opennetworking.org/.

Frank, A. (2010). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science. *http://archive. ics. uci. edu/ml*.

Fregnan, E., Petrulio, F. & Bacchelli, A. (2022). The evolution of the code during review: an investigation on review changes. *Empirical Software Engineering*, 27(7), 177.

Freitas, A. A. (2014). Comprehensible classification models: a position paper. *ACM SIGKDD explorations newsletter*, 15(1), 1–10.

Friedler, S., Scheidegger, C. & Venkatasubramanian, S. (2014). Certifying and removing disparate impact. *arXiv preprint arXiv:1412.3756*.

Fuggetta, A. (2000). Software process: a roadmap. *Proceedings of the Conference on the Future of Software Engineering*, pp. 25–34.

Gamma, E., Vlissides, J., Helm, R., Johnson, R. & Ralph, J. (1995). *Design patterns: elements of reusable object-oriented software*. Reading, Mass: Addison-Wesley.

Gao, H., Lu, M., Pan, C. & Xu, B. (2019). Empirical Study: Are Complex Network Features Suitable for Cross-Version Software Defect Prediction? *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*, pp. 1–5.

Garcia, S., Luengo, J., Sáez, J. A., Lopez, V. & Herrera, F. (2012). A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning. *IEEE transactions on Knowledge and Data Engineering*, 25(4), 734–750.

Gay, G., Menzies, T., Davies, M. & Gundy-Burlet, K. (2010). Automatically finding the control variables for complex system behavior. *Automated Software Engineering*, 17, 439–468.

Ge, J., Liu, J. & Liu, W. (2018). Comparative study on defect prediction algorithms of supervised learning software based on imbalanced classification data sets. *2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pp. 399–406.

Gember, A., Prabhu, P., Ghadiyali, Z. & Akella, A. (2012). Toward software-defined middlebox networking. *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pp. 7–12.

Ghaleb, T., Yu, Y. & Briand, L. C. (2019). An Empirical Study of the Effectiveness of Different Features for Predicting Build Outcome. *Empirical Software Engineering*, 24, 2800–2838. doi: 10.1007/s10664-019-09718-9.

Ghosh, D. & Vogt, A. (2012). Outliers: An evaluation of methodologies. *Joint statistical meetings*, 12(1), 3455–3460.

Ghotra, B., McIntosh, S. & Hassan, A. E. (2015). Revisiting the impact of classification techniques on the performance of defect prediction models. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, 1, 789–800.

Ghotra, B., McIntosh, S. & Hassan, A. E. (2017). A large-scale study of the impact of feature selection techniques on defect classification models. *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pp. 146–157.

GitHub. [API version 2022-11-28]. (2024). GitHub REST API: Pull Requests. Retrieved from: https://docs.github.com/en/rest/pulls/pulls.

GitLab. (2024). GitLab Documentation: Associated Records and Ghost User. Retrieved from: https://docs.gitlab.com/user/profile/account/delete_account/#associated-records.

Golder, S. & Huberman, B. A. (2005). The structure of collaborative tagging systems. *arXiv preprint cs/0508082*.

Golzadeh, M., Decan, A., Legay, D. & Mens, T. (2021). A ground-truth dataset and classification model for detecting bots in GitHub issue and PR comments. *Journal of Systems and Software*, 175, 110911.

Gong, A., Yang, K., Lyu, J. & Li, X. (2024). A two-stage reinforcement learning-based approach for multi-entity task allocation. *Engineering Applications of Artificial Intelligence*, 136, 108906. doi: 10.1016/j.engappai.2024.108906.

Gong, L., Jiang, S. & Jiang, L. (2019). Tackling class imbalance problem in software defect prediction through cluster-based over-sampling with filtering. *IEEE Access*, 7, 145725–145737.

Gong, L., Rajbahadur, G. K., Hassan, A. E. & Jiang, S. (2021). Revisiting the impact of dependency network metrics on software defect prediction. *IEEE Transactions on Software Engineering*, 48(12), 5030–5049.

Goswami, I. & Urminsky, O. (2020). More time, more work: How time limits bias estimates of task scope and project duration. *Judgment and Decision Making*, 15(6), 994–1008. doi: 10.1017/S1930297500008196.

Gousios, G. & Spinellis, D. (2017). Mining software engineering data from GitHub. *2017 IEEE/ACM 39th international conference on software engineering companion (ICSE-C)*, pp. 501–502.

Gousios, G., Vasilescu, B., Serebrenik, A. & Zaidman, A. (2014). Lean GHTorrent: GitHub data on demand. *Proceedings of the 11th working conference on mining software repositories*, pp. 384–387.

Gousios, G., Zaidman, A., Storey, M.-A. & van Deursen, A. (2015). Work Practices and Challenges in Pull-Based Development: The Contributor's Perspective. *ICSE*, pp. 285–296. doi: 10.1109/ICSE.2015.45.

Greenwade, G. D. (1993). The Comprehensive Tex Archive Network (CTAN). *TUGBoat*, 14(3), 342–351.

Greiler, M., Bird, C., Storey, M.-A., MacLeod, L. & Czerwonka, J. (2016). Code reviewing in the trenches: Understanding challenges, best practices and tool needs.

Gress, T. W., Denvir, J. & Shapiro, J. I. (2018). Effect of removing outliers on statistical inference: implications to interpretation of experimental data in medical research. *Marshall journal of medicine*, 4(2).

Gu, Q., Zhu, L. & Cai, Z. (2009). Evaluation measures of the classification performance of imbalanced data sets. *Computational Intelligence and Intelligent Systems: 4th International Symposium, ISICA 2009, Huangshi, China, October 23-25, 2009. Proceedings 4*, pp. 461–471.

Guo, P. J., Zimmermann, T., Nagappan, N. & Murphy, B. (2010). Characterizing and predicting which bugs get fixed: an empirical study of microsoft windows. *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pp. 495–504.

Gupta, A. & Sundaresan, N. (2018). Intelligent code reviews using deep learning. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'18) Deep Learning Day*.

Gupta, N., Mujumdar, S., Patel, H., Masuda, S., Panwar, N., Bandyopadhyay, S., Mehta, S., Guttula, S., Afzal, S., Sharma Mittal, R. et al. (2021). Data quality for machine learning tasks. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 4040–4041.

Gupta, R., Pal, S., Kanade, A. & Shevade, S. (2017). Deepfix: Fixing common c language errors by deep learning. *Proceedings of the aaai conference on artificial intelligence*, 31(1).

Gupta, R., Tanwar, S., Tyagi, S. & Kumar, N. (2020). Machine learning models for secure data analytics: A taxonomy and threat model. *Computer Communications*, 153, 406–440.

Gupta, S. & Gupta, A. (2019). Dealing with noise problem in machine learning data-sets: A systematic review. *Procedia Computer Science*, 161, 466–474.

Gupta, S., Sikka, G. & Verma, H. (2011). Recent methods for software effort estimation by analogy. *ACM SIGSOFT Software Engineering Notes*, 36(4), 1–5. doi: 10.1145/1988997.1989016.

Haider, M. A., Mostofa, A. B., Mosaddek, S. S. B., Iqbal, A. & Ahmed, T. (2024). Prompting and Fine-tuning Large Language Models for Automated Code Review Comment Generation. *arXiv preprint arXiv:2411.10129*.

Halimu, C., Kasem, A. & Newaz, S. S. (2019). Empirical comparison of area under ROC curve (AUC) and Mathew correlation coefficient (MCC) for evaluating machine learning algorithms on imbalanced datasets for binary classification. *Proceedings of the 3rd international conference on machine learning and soft computing*, pp. 1–6.

Hardt, M., Price, E., Srebro, N. et al. (2016). Equality of opportunity in supervised learning, in 'Advances in neural information processing systems'.

Hasan, K. A., Macedo, M., Tian, Y., Adams, B. & Ding, S. (2023). Understanding the Time to First Response In GitHub Pull Requests. *arXiv preprint arXiv:2304.08426*.

Hasan, M., Iqbal, A., Islam, M. R. U., Rahman, A. I. & Bosu, A. (2021). Using a balanced scorecard to identify opportunities to improve code review effectiveness: An industrial experience report. *Empirical Software Engineering*, 26, 1–34.

Hassan, S., Bezemer, C.-P. & Hassan, A. E. (2018). Studying bad updates of top free-to-download apps in the google play store. *IEEE Transactions on Software Engineering*, 46(7), 773–793.

Hemández-Molinos, M. J., Sánchez-García, Á. J. & Barrientos-Martínez, R. E. (2021). Classification Algorithms for Software Defect Prediction: A Systematic Literature Review. *2021 9th International Conference in Software Engineering Research and Innovation (CONISOFT)*, pp. 189–196.

Herbold, S. (2019). On the costs and profit of software defect prediction. *IEEE Transactions on Software Engineering*, 47(11), 2617–2631.

Herbold, S., Trautsch, A. & Grabowski, J. (2017). Global vs. local models for cross-project defect prediction. *Empirical software engineering*, 22(4), 1866–1902.

Ho, T. K. & Basu, M. (2002). Complexity measures of supervised classification problems. *IEEE transactions on pattern analysis and machine intelligence*, 24(3), 289–300.

Hollenbeck, J. R., DeRue, D. S. & Mannor, M. (2006). Statistical power and parameter stability when subjects are few and tests are many: Comment on Peterson, Smith, Martorana, and Owens (2003).

Hossin, M., Sulaiman, M., Mustapha, A., Mustapha, N. & Rahmat, R. (2011). A hybrid evaluation metric for optimizing classifier. *2011 3rd Conference on Data Mining and Optimization (DMO)*, pp. 165–170.

Hossin, M. & Sulaiman, M. N. (2015). A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process*, 5(2), 1.

Huang, J. & Ling, C. X. (2005). Using AUC and accuracy in evaluating learning algorithms. *IEEE Transactions on knowledge and Data Engineering*, 17(3), 299–310.

Huang, Y., Jia, N., Zhou, X., Hong, K. & Chen, X. (2020). Would the patch be quickly merged? *Blockchain and Trustworthy Systems: First International Conference, BlockSys 2019, Guangzhou, China, December 7–8, 2019, Proceedings 1*, pp. 461–475.

Huang, Z. (2010). *Automatically identifying configuration files*. (Ph.D. thesis).

Humble, J. & Farley, D. (2010). *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education.

Husein, S. & Oxley, A. (2009). A coupling and cohesion metrics suite for object-oriented software. *2009 International Conference on Computer Technology and Development*, 1, 421–425.

Iden, J., Tessem, B. & Päivärinta, T. (2011). Problems in the interplay of development and IT operations in system development projects: A Delphi study of Norwegian IT experts. *Information and Software Technology*, 53(4), 394–406.

Ikonen, M., Kettunen, P., Oza, N. & Abrahamsson, P. (2010). Exploring the Sources of Waste in Kanban Software Development Projects. *2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications*, pp. 376-381. doi: 10.1109/SEAA.2010.40.

Imtiaz, S. & Ikram, N. (2017). Dynamics of task allocation in global software development. *Journal of Software: Evolution and Process*, 29(1), e1832. doi: 10.1002/smr.1832. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.1832.

Islam, K., Ahmed, T., Shahriyar, R., Iqbal, A. & Uddin, G. (2022). Early prediction for merged vs abandoned code changes in modern code reviews. *Information and Software Technology*, 142, 106756.

Jaccard, P. (1901). Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bull Soc Vaudoise Sci Nat*, 37, 547–579.

Jalali, O., Menzies, T. & Feather, M. (2008). Optimizing requirements decisions with keys. *Proceedings of the 4th international workshop on Predictor models in software engineering*, pp. 79–86.

Jaoua, I., Sghaier, O. B. & Sahraoui, H. (2025). Combining Large Language Models with Static Analyzers for Code Review Generation. *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*, pp. 174–186.

Jiang, W., Zhang, X., Xie, X., Yu, J., Zhi, Y., Ma, S. & Shen, C. (2025). The Foundation Cracks: A Comprehensive Study on Bugs and Testing Practices in LLM Libraries. *arXiv preprint arXiv:2506.12320*.

Jiang, Y., Adams, B. & German, D. M. (2013). Will my patch make it? and how fast? case study on the linux kernel. *2013 10th Working Conference on Mining Software Repositories (MSR)*, pp. 101–110.

Jiarpakdee, J., Tantithamthavorn, C., Ihara, A. & Matsumoto, K. (2016). A study of redundant metrics in defect prediction datasets. *2016 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 51–52.

Jiarpakdee, J., Tantithamthavorn, C. & Hassan, A. E. (2019). The impact of correlated metrics on the interpretation of defect models. *IEEE Transactions on Software Engineering*, 47(2), 320–331.

Jiarpakdee, J., Tantithamthavorn, C. & Treude, C. (2020). The impact of automated feature selection techniques on the interpretation of defect models. *Empirical Software Engineering*, 25(5), 3590–3638.

Jing, X.-Y., Wu, F., Dong, X. & Xu, B. (2016). An improved SDA based defect prediction framework for both within-project and cross-project class-imbalance problems. *IEEE Transactions on Software Engineering*, 43(4), 321–339.

Jørgensen, M. & Sjøberg, D. I. K. (2001). Impact of Effort Estimates on Software Project Work. *Information and Software Technology*, 43(15), 939–948. doi: 10.1016/S0950-5849(01)00186-1.

Jureczko, M., Kajda, Ł. & Górecki, P. (2020). Code review effectiveness: an empirical study on selected factors influence. *IET Software*, 14(7), 794–805.

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M. & Damian, D. (2014). The promises and perils of mining GitHub. *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR)*, pp. 92–101.

Kamei, Y., Monden, A., Matsumoto, S., Kakimoto, T. & Matsumoto, K.-i. (2007). The effects of over and under sampling on fault-prone module detection. *First international symposium on empirical software engineering and measurement (ESEM 2007)*, pp. 196–204.

Kanban University. [Compiled by Susanne Bartel and Andreas Bartel with the collaboration of the Kanban University team]. (2022). The Official Guide to The Kanban Method. Retrieved from: https://kanban.university/wp-content/uploads/2023/04/The-Official-Kanban-Guide_A4.pdf.

Kansab, S., Bordeleau, F. & Tizghadam, A. (2025a). On The Impact of Merge Request Deviations on Code Review Practices. *arXiv preprint arXiv:2506.08860*.

Kansab, S., Cherguelaine, O., Merabet, M. R., Bordeleau, F. & Tizghadam, A. (2025b). RevMine: An LLM-Assisted Tool for Code Review Mining and Analysis Across Git Platforms. *Proceedings of the 35th IEEE International Conference on Collaborative Advances in Software and Computing (CASCON)*.

Kansab, S., Hanania, M., Bordeleau, F., Tizghadam, A. et al. (2025c). Leveraging Merge Request Data to Analyze Devops Practices: Insights From a Networking Software Solution Company. *CS & IT Conference Proceedings*, 15(10).

Kansab, S., Sayagh, M., Bordeleau, F. & Tizghadam, A. (2025d). An Empirical Study on the Amount of Changes Required for Merge Request Acceptance. *arXiv preprint arXiv:2507.23640*.

Karamcheti, S., Krishna, R., Fei-Fei, L. & Manning, C. D. (2021). Mind your outliers! investigating the negative impact of outliers on active learning for visual question answering. *arXiv preprint arXiv:2107.02331*.

Kawata, K., Amasaki, S. & Yokogawa, T. (2015). Improving relevancy filter methods for cross-project defect prediction. *2015 3rd International Conference on Applied Computing and Information Technology/2nd International Conference on Computational Science and Intelligence*, pp. 2–7.

Khan, A. & Zubair, S. (2020). Expansion of regularized kmeans discretization machine learning approach in prognosis of dementia progression. *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pp. 1–6.

Khatoonabadi, S., Costa, D. E., Abdalkareem, R. & Shihab, E. (2023). On Wasted Contributions: Understanding the Dynamics of Contributor-Abandoned Pull Requests–A Mixed-Methods Study of 10 Large Open-Source Projects. *ACM Transactions on Software Engineering and Methodology*, 32(1), 1–39.

Khatoonabadi, S., Abdellatif, A., Costa, D. E. & Shihab, E. (2024). Predicting the first response latency of maintainers and contributors in pull requests. *IEEE Transactions on Software Engineering*, 50(10), 2529–2543.

Kim, G., Humble, J., Debois, P., Willis, J., Forsgren, N. & Allspaw, J. (2021). *The DevOps handbook : how to create world-class agility, reliability, & security in technology organizations* (ed. Second edition). Portland, OR: IT Revolution Press.

Kim, H., Kwon, Y., Joh, S., Kwon, H., Ryou, Y. & Kim, T. (2022). Understanding automated code review process and developer experience in industry. *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1398–1407.

Kim, S., Zimmermann, T., Pan, K., James Jr, E. et al. (2006). Automatic identification of bug-introducing changes. *21st IEEE/ACM international conference on automated software engineering (ASE'06)*, pp. 81–90.

Kim, S., Zimmermann, T., Whitehead Jr, E. J. & Zeller, A. (2007). Predicting faults from cached history. *29th International Conference on Software Engineering (ICSE'07)*, pp. 489–498.

Kim, S., Zhang, H., Wu, R. & Gong, L. (2011). Dealing with noise in defect prediction. *Proceedings of the 33rd International Conference on Software Engineering*, pp. 481–490.

Kim, Y., Mun, S., Yoo, S. & Kim, M. (2019). Precise learn-to-rank fault localization using dynamic and static features of target programs. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 28(4), 1–34.

Kitchenham, B., Pfleeger, S. L. & Fenton, N. (1995). Towards a framework for software measurement validation. *IEEE Transactions on software Engineering*, 21(12), 929–944.

Kitchenham, B., Madeyski, L., Budgen, D., Keung, J., Brereton, P., Charters, S., Gibbs, S. & Pohthong, A. (2017). Robust statistical methods for empirical software engineering. *Empirical Software Engineering*, 22, 579–630.

Klein, B., Tyler, C. & Fields, S. (2022). DevOps and Data: Faster-Time-to-Knowledge through SageOps, MLOps, and DataOps.

Klösgen, W. (1996). Explora: A multipattern and multistrategy discovery assistant. In *Advances in knowledge discovery and data mining* (pp. 249–271).

Kondo, M., Bezemer, C.-P., Kamei, Y., Hassan, A. E. & Mizuno, O. (2019). The impact of feature reduction techniques on defect prediction models. *Empirical Software Engineering*, 24, 1925–1963.

Kononenko, O., Baysal, O. & Godfrey, M. W. (2016). Code review quality: How developers see it. *Proceedings of the 38th international conference on software engineering*, pp. 1028–1038.

Konukoglu, E. & Ganz, M. (2014). Approximate false positive rate control in selection frequency for random forest. *arXiv preprint arXiv:1410.2838*.

Koponen, T., Shenker, S., Balakrishnan, H., Feamster, N., Ganichev, I., Ghodsi, A., Godfrey, P. B., McKeown, N., Parulkar, G., Raghavan, B. et al. (2011). Architecting for innovation. *ACM SIGCOMM Computer Communication Review*, 41(3), 24–36.

Kraska, T., Talwalkar, A., Duchi, J. C., Griffith, R., Franklin, M. J. & Jordan, M. I. (2013). MLbase: A Distributed Machine-learning System. *Cidr*, 1, 2–1.

Kraut, R. E. & Streeter, L. A. (1995). Coordination in software development. *Commun. ACM*, 38(3), 69–81. doi: 10.1145/203330.203345.

Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S. & Uhlig, S. (2014). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1), 14–76.

Krishnan Rajbahadur, G., Wang, S., Kamei, Y. & Hassan, A. E. (2022). The impact of feature importance methods on the interpretation of defect classifiers. *arXiv e-prints*, arXiv–2202.

Krutauz, A., Dey, T., Rigby, P. C. & Mockus, A. (2020). Do code review measures explain the incidence of post-release defects? Case study replications and bayesian networks. *Empirical Software Engineering*, 25, 3323–3356.

Kubat, M., Holte, R. & Matwin, S. (1997). Learning when negative examples abound. *European conference on machine learning*, pp. 146–153.

Kumeno, F. (2019). Sofware engneering challenges for machine learning applications: A literature review. *Intelligent Decision Technologies*, 13(4), 463–476.

Kuznetsova, A., Brockhoff, P. B. & Christensen, R. H. (2017). lmerTest package: tests in linear mixed effects models. *Journal of statistical software*, 82, 1–26.

Lal, H. & Pahwa, G. (2017). Code review analysis of software system using machine learning techniques. *2017 11th International Conference on Intelligent Systems and Control (ISCO)*, pp. 8–13.

Lamersdorf, A., Munch, J. & Rombach, D. (2009). A Survey on the State of the Practice in Distributed Software Development: Criteria for Task Allocation. *2009 Fourth IEEE International Conference on Global Software Engineering*, pp. 41–50. doi: 10.1109/ICGSE.2009.12.

Lavrac, N., Kavsek, B., Flach, P. & Todorovski, L. (2004). Subgroup discovery with CN2-SD. *J. Mach. Learn. Res.*, 5(2), 153–188.

Le Cessie, S. & Van Houwelingen, J. C. (1992). Ridge estimators in logistic regression. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 41(1), 191–201.

Lee, D., Rajbahadur, G. K., Lin, D., Sayagh, M., Bezemer, C.-P. & Hassan, A. E. (2020). An empirical study of the characteristics of popular Minecraft mods. *Empirical Software Engineering*, 25, 3396–3429.

Legault, J. (2022). *An empirical investigation on the prevalence, impact, and manifestation of rework commits in the merge requests mechanism*. (Ph.D. thesis, École de technologie supérieure).

Li, H. B., Wang, W., Ding, H. W. & Dong, J. (2010). Trees weighting random forest method for classifying high-dimensional noisy data. *2010 IEEE 7th international conference on e-business engineering*, pp. 160–163.

Li, L., Yang, L., Jiang, H., Yan, J., Luo, T., Hua, Z., Liang, G. & Zuo, C. (2022a). AUGER: automatically generating review comments with pre-training models. *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1009–1021.

Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J. & Su, B.-Y. (2014). Scaling distributed machine learning with the parameter server. *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pp. 583–598.

Li, Z., Zhang, X., Guo, J. & Shang, Y. (2019). Class imbalance data-generation for software defect prediction. *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*, pp. 276–283.

Li, Z., Zhang, H., Jing, X.-Y., Xie, J., Guo, M. & Ren, J. (2022b). DSSDPP: Data Selection and Sampling Based Domain Programming Predictor for Cross-Project Defect Prediction. *IEEE Transactions on Software Engineering*.

Li, Z., Yu, Y., Zhou, M., Wang, T., Yin, G., Lan, L. & Wang, H. (2020). Redundancy, context, and preference: An empirical study of duplicate pull requests in OSS projects. *IEEE Transactions on Software Engineering*, 48(4), 1309–1335.

Liang, Q., Sun, Z., Zhu, Q., Hu, J., Zhao, Y. & Zhang, L. (2023). Cupcleaner: A data cleaning approach for comment updating. *arXiv preprint arXiv:2308.06898*.

Liao, H., Li, Y. & Brooks, G. (2016). Outlier impact and accommodation methods: Multiple comparisons of Type I error rates. *Journal of Modern Applied Statistical Methods*, 15(1), 23.

Lin, C. & Snyder, L. (2008). *Principles of Parallel Programming* (ed. 1st). USA: Addison-Wesley Publishing Company.

Lin, D., Tantithamthavorn, C. & Hassan, A. E. (2021). The impact of data merging on the interpretation of cross-project just-in-time defect models. *IEEE Transactions on Software Engineering*.

Lin, J. (2013). Context-aware task allocation for distributed agile team. *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 758–761. doi: 10.1109/ASE.2013.6693151.

Liu, C., Lin, H. Y. & Thongtanunam, P. (2025). Too Noisy To Learn: Enhancing Data Quality for Code Review C. *arXiv preprint arXiv:2502.02757*.

Liu, F., Kibalya, G., Santhosh Kumar, S. & Zhang, P. (2021). Challenges of traditional networks and development of programmable networks. In *Software defined internet of everything* (pp. 37–61). Springer.

Liu, Y., Zhang, W., Qin, G. & Zhao, J. (2022). A comparative study on the effect of data imbalance on software defect prediction. *Procedia Computer Science*, 214, 1603–1616.

Lo, D., Scanniello, G., Marchetto, A., Ali, N. & McMillan, C. (2014). Leveraging machine learning and information retrieval techniques in software evolution tasks: summary of the first MALIR-SE workshop, at ASE 2013. *ACM SIGSOFT Software Engineering Notes*, 39(1), 1–2.

Lomio, F., Moreschini, S. & Lenarduzzi, V. (2022). A machine and deep learning analysis among sonarqube rules, product, and process metrics for faults prediction.

Lounis, H., Gayed, T. F. & Boukadoum, M. (2011). Machine-learning models for software quality: a compromise between performance and intelligibility. *2011 IEEE 23rd International Conference on Tools with Artificial Intelligence*, pp. 919–921.

Lukins, S. K., Kraft, N. A. & Etzkorn, L. H. (2008). Source code retrieval for bug localization using latent dirichlet allocation. *2008 15Th working conference on reverse engineering*, pp. 155–164.

Macgregor, G. & McCulloch, E. (2006). Collaborative tagging as a knowledge organisation and resource discovery tool. *Library review*, 55(5), 291–300.

Maddila, C., Upadrasta, S. S., Bansal, C., Nagappan, N., Gousios, G. & van Deursen, A. (2023). Nudge: Accelerating Overdue Pull Requests toward Completion. *ACM Transactions on Software Engineering and Methodology*, 32(2), 1–30.

Mahmood, S., Anwer, S., Niazi, M., Alshayeb, M. & Richardson, I. (2017). Key factors that influence task allocation in global software development. *Information and Software Technology*, 91, 102–122. doi: 10.1016/j.infsof.2017.06.009.

Mahmood, Y., Kama, N., Azmi, A., Khan, A. S. & Ali, M. (2022). Software effort estimation accuracy prediction of machine learning techniques: A systematic performance evaluation. *Software: Practice and experience*, 52(1), 39–65.

Maiga, S., Bilgaiyan, S. & Sagnika, S. (2025). Predicting software effort using BERT-based word embeddings. *International Journal of System Assurance Engineering and Management*, 16(5), 1728–1742.

Mallidi, R. K. & Sharma, M. (2021a). Study on agile story point estimation techniques and challenges. *Int. J. Comput. Appl*, 174(13), 9–14.

Mallidi, R. K. & Sharma, M. (2021b). Study on agile story point estimation techniques and challenges. *Int. J. Comput. Appl*, 174(13), 9–14.

Marcus, A., Sergeyev, A., Rajlich, V. & Maletic, J. I. (2004). An information retrieval approach to concept location in source code. *11th working conference on reverse engineering*, pp. 214–223.

Mateen, M. A. & Malik, A. A. (2023). A comparative study of the accuracy and efficiency of wideband delphi and planning poker software effort estimation techniques. *2023 International Conference on IT and Industrial Technologies (ICIT)*, pp. 1–5.

Mayez, M., Nagaty, K. & Hamdy, A. [arXiv:2202.01713]. (2022). Developer Load Normalization Using Iterative Kuhn-Munkres Algorithm: An Optimization Triaging Approach. arXiv. Retrieved on 2024-10-15 from: http://arxiv.org/abs/2202.01713.

Mazzara, M. (2026). Software Release anomaly detection in DevOps environment.

McIntosh, S., Kamei, Y., Adams, B. & Hassan, A. E. (2014). The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. *Proceedings of the 11th working conference on mining software repositories*, pp. 192–201.

McIntosh, S., Kamei, Y., Adams, B. & Hassan, A. E. (2016a). An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering*, 21, 2146–2189.

McIntosh, S., Kamei, Y., Adams, B. & Hassan, A. E. (2016b). An Empirical Study of the Impact of Modern Code Review Practices on Software Quality. *Empirical Software Engineering*, 21(5), 2146–2189. doi: 10.1007/s10664-015-9380-8.

McMillan, S. S., King, M. & Tully, M. P. (2016). How to use the nominal group and Delphi techniques. *International Journal of Clinical Pharmacy*, 38(3), 655–662.

Menzies, T., Butcher, A., Cok, D., Marcus, A., Layman, L., Shull, F., Turhan, B. & Zimmermann, T. (2012). Local versus global lessons for defect prediction and effort estimation. *IEEE Transactions on software engineering*, 39(6), 822–834.

Miranda, D., Palma, D., Fernández, A., Noel, R., Cechinel, C. & Munoz, R. (2024). Enhancing Agile Project Management Education with AI: ChatGPT-4's Role in Evaluating Student Contributions. *2024 43rd International Conference of the Chilean Computer Science Society (SCCC)*, pp. 1–4.

Monalisa, S. & Kurnia, F. (2019). Analysis of DBSCAN and K-means algorithm for evaluating outlier on RFM model of customer behaviour. *Telkomnika (Telecommunication Computing Electronics and Control)*, 17(1), 110–117.

Mori, T. & Uchihira, N. (2019). Balancing the trade-off between accuracy and interpretability in software defect prediction. *Empirical Software Engineering*, 24(2), 779–825.

Moser, R., Pedrycz, W. & Succi, G. (2008). A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. *Proceedings of the 30th international conference on Software engineering*, pp. 181–190.

Mossman, D. (1994). Assessing predictions of violence: being accurate about accuracy. *Journal of consulting and clinical psychology*, 62(4), 783.

Mourão-Miranda, J., Hardoon, D. R., Hahn, T., Marquand, A. F., Williams, S. C., Shawe-Taylor, J. & Brammer, M. (2011). Patient classification as an outlier detection problem: an application of the one-class support vector machine. *Neuroimage*, 58(3), 793–804.

Mukadam, M., Bird, C. & Rigby, P. C. (2013). Gerrit software code review data from android. *2013 10th Working Conference on Mining Software Repositories (MSR)*, pp. 45–48.

Müller, H., Kharitonov, A., Nahhas, A., Bosse, S. & Turowski, K. (2022). Addressing IT Capacity Management Concerns Using Machine Learning Techniques. *SN Computer Science*, 3, 1–15.

Myllyaho, L., Raatikainen, M., Männistö, T., Mikkonen, T. & Nurminen, J. K. (2021). Systematic literature review of validation methods for AI systems. *Journal of Systems and Software*, 181, 111050.

Nagappan, N., Murphy, B. & Basili, V. (2008). The influence of organizational structure on software quality: an empirical case study. *Proceedings of the 30th international conference on Software engineering*, (ICSE '08), 521–530. doi: 10.1145/1368088.1368160.

Nakra, V., Dave, A., Chenchala, P. K. & Agarwal, A. (2023). Enhancing Software Project Management and Task Allocation with AI and Machine Learning. *International Journal on Recent and Innovation Trends in Computing and Communication*, 11, 1171–1178.

Nam, J., Pan, S. J. & Kim, S. (2013). Transfer defect learning. *2013 35th international conference on software engineering (ICSE)*, pp. 382–391.

Nejati, M., Alfadel, M. & McIntosh, S. (2023). Code review of build system specifications: prevalence, purposes, patterns, and perceptions. *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pp. 1213–1224.

Neto, E. C., Da Costa, D. A. & Kulesza, U. (2018). The impact of refactoring changes on the szz algorithm: An empirical study. *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 380–390.

Nguyen, P. T., Di Rocco, J., Di Sipio, C., Shakya, M., Di Ruscio, D. & Di Penta, M. (2024). Automatic Categorization of GitHub Actions with Transformers and Few-shot Learning. *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 468–474.

Nizam, A. (2022). Software project failure process definition. *IEEE access*, 10, 34428–34441.

Nogueira, A. F., Ribeiro, J. C., Zenha-Rela, M. A. & Craske, A. (2018). Improving la redoute's ci/cd pipeline and devops processes by applying machine learning techniques. *2018 11th international conference on the quality of information and communications technology (QUATIC)*, pp. 282–286.

Notaro, P., Cardoso, J. & Gerndt, M. (2020). A systematic mapping study in AIOps. *International Conference on Service-Oriented Computing*, pp. 110–123.

Nundlall, C. & Nagowah, S. (2021). Task allocation and coordination in distributed agile software development: a systematic review. *International Journal of Information Technology*, 13(1), 321–30. doi: 10.1007/s41870-020-00543-4. Place: Germany Publisher: Springer.

Nunes, B. A. A., Mendonca, M., Nguyen, X.-N., Obraczka, K. & Turletti, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications surveys & tutorials*, 16(3), 1617–1634.

Oh, J.-S. & Choi, H.-J. (2005). A reflective practice of automated and manual code reviews for a studio project. *Fourth Annual ACIS International Conference on Computer and Information Science (ICIS'05)*, pp. 37–42.

Olewicki, D., Habchi, S. & Adams, B. (2024a). An Empirical Study on Code Review Activity Prediction and Its Impact in Practice. *Proceedings of the ACM on Software Engineering*. doi: 10.1145/3663529. Presented at FSE 2024.

Olewicki, D., Habchi, S. & Adams, B. (2024b). An empirical study on code review activity prediction and its impact in practice. *Proceedings of the ACM on Software Engineering*, 1(FSE), 2238–2260.

Ollama. (2024a). Ollama: Get up and running with large language models locally. Retrieved from: https://ollama.com/.

Ollama. (2024b). Ollama: OpenAI Compatibility. Retrieved from: https://github.com/ollama/ollama/blob/main/docs/openai.md.

OpenAI. (2025). https://chatgpt.com/ [Large Language Model (LLM)]. Retrieved on 2025-01-15 from: https://chatgpt.com/.

OpenRouter. (2024a). OpenRouter: A unified API for LLMs. Retrieved from: https://openrouter.ai/.

OpenRouter. (2024b). OpenRouter: Programming/Coding Models. Retrieved from: https://openrouter.ai/models?category=programming.

OpenRouter. (2024c). OpenRouter: Free Models. Retrieved from: https://openrouter.ai/models?max_price=0.

OpenRouter. (2024d). OpenRouter: Models Catalogue. Retrieved from: https://openrouter.ai/models.

Ord, K. et al. (1996). Outliers in statistical data: V. Barnett and T. Lewis, 1994, (John Wiley & Sons, Chichester), 584 pp.,[UK pound] 55.00, ISBN 0-471-93094-6. *International Journal of Forecasting*, 12(1), 175–176.

Osborne, J. W. & Overbay, A. (2004). The power of outliers (and why researchers should always check for them). *Practical Assessment, Research, and Evaluation*, 9(1), 6.

Ouatiti, Y. E., Sayagh, M., Kerzazi, N. & Hassan, A. E. (2022). An empirical study on log level prediction for multi-component systems. *IEEE Transactions on Software Engineering*, (01), 1–1.

Panichella, A., Oliveto, R. & De Lucia, A. (2014). Cross-project defect prediction models: L'union fait la force. *2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, pp. 164–173.

Pascarella, L., Spadini, D., Palomba, F., Bruntink, M. & Bacchelli, A. (2018). Information needs in contemporary code review. *Proceedings of the ACM on human-computer interaction*, 2(CSCW), 1–27.

Pasuksmit, J., Thongtanunam, P. & Karunasekera, S. (2024). A systematic literature review on reasons and approaches for accurate effort estimations in agile. *ACM Computing Surveys*, 56(11), 1–37.

Pereira, S., Meier, R., McKinley, R., Wiest, R., Alves, V., Silva, C. A. & Reyes, M. (2018). Enhancing interpretability of automatically extracted machine learning features: application to a RBM-Random Forest system on brain lesion segmentation. *Medical image analysis*, 44, 228–244.

Pérez-Verdejo, J. M., Sánchez-García, A. J. & Ocharán-Hernández, J. O. (2020). A systematic literature review on machine learning for automated requirements classification. *2020 8th International Conference in Software Engineering Research and Innovation (CONISOFT)*, pp. 21–28.

Petrić, J., Bowes, D., Hall, T., Christianson, B. & Baddoo, N. (2016). Building an ensemble for software defect prediction based on diversity selection. *Proceedings of the 10th ACM/IEEE International symposium on empirical software engineering and measurement*, pp. 1–10.

Pirouzkhah, S., Gonçalves, P. W. & Bacchelli, A. (2026). The Value of Effective Pull Request Description. *Proceedings of the 23rd International Conference on Mining Software Repositories (MSR)*. doi: 10.1145/3793302.3793368.

Poppendieck, M. & Cusumano, M. A. (2012). Lean Software Development: A Tutorial. *IEEE Software*, 29(5), 26–32. doi: 10.1109/MS.2012.107. Conference Name: IEEE Software.

Pornprasit, C. & Tantithamthavorn, C. (2024). Fine-tuning and prompt engineering for large language models-based code review automation. *Information and Software Technology*, 175, 107523.

Potvin, R. & Levenberg, J. (2016). Why Google stores billions of lines of code in a single repository. *Communications of the ACM*, 59(7), 78–87.

Poženel, M., Fürst, L., Vavpotič, D. & Hovelja, T. (2023). Agile effort estimation: Comparing the accuracy and efficiency of planning poker, bucket system, and affinity estimation methods. *International Journal of Software Engineering and Knowledge Engineering*, 33(11n12), 1923–1950.

Prasad, A. M., Iverson, L. R. & Liaw, A. (2006). Newer classification and regression tree techniques: bagging and random forests for ecological prediction. *Ecosystems*, 9, 181–199.

Qian, L., Yao, Q., Khoshgoftaar, T. M. et al. (2010). Dynamic Two-phase Truncated Rayleigh Model for Release Date Prediction of Software. *Journal of Software Engineering and Applications*, 3(06), 603.

R-bloggers. [[Accessed 16-Mar-2023]]. (2023). Kendall's Rank Correlation in R-Correlation Test — r-bloggers.com. Retrieved from: https://www.r-bloggers.com/2021/06/kendalls-rank-correlation-in-r-correlation-test/.

Radliński, Ł. (2024). The Trade-off Between Data Volume and Quality in Predicting User Satisfaction in Software Projects. *2024 50th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 483–490.

Radliński, Ł. & Swacha, J. (2025). Large Language Models for Early-Stage Software Project Estimation: A Systematic Mapping Study. *Applied Sciences*, 15(24), 13099.

Rahman, A., Stallings, J. & Williams, L. (2018). Defect prediction metrics for infrastructure as code scripts in DevOps. *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, pp. 414–415.

Rahman, A., Rahman, M. R., Parnin, C. & Williams, L. (2021). Security smells in ansible and chef scripts: A replication study. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 30(1), 1–31.

Rahman, M. M., Roy, C. K. & Kazman, R. (2016). CoRReCT: Code Reviewer Recommendation in GitHub Based on Cross-Project and Technology Experience. *Proceedings of the 38th International Conference on Software Engineering Companion (ICSE-C)*, pp. 222–231. doi: 10.1145/2889160.2889244.

Rahman, M. M., Roy, C. K. & Kula, R. G. (2017). Predicting usefulness of code review comments using textual features and developer experience. *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pp. 215–226.

Rajbahadur, G. K., Wang, S., Kamei, Y. & Hassan, A. E. (2017). The impact of using regression models to build defect classifiers. *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pp. 135–145.

Rajbahadur, G. K., Wang, S., Kamei, Y. & Hassan, A. E. (2019). Impact of discretization noise of the dependent variable on machine learning classifiers in software engineering. *IEEE Transactions on Software Engineering*, 47(7), 1414–1430.

Rajbahadur, G. K., Wang, S., Oliva, G. A., Kamei, Y. & Hassan, A. E. (2021). The impact of feature importance methods on the interpretation of defect classifiers. *IEEE Transactions on Software Engineering*, 48(7), 2245–2261.

Rana, R., Staron, M., Berger, C., Hansson, J., Nilsson, M. & Meding, W. (2014). The adoption of machine learning techniques for software defect prediction: An initial industrial validation. *Knowledge-Based Software Engineering: 11th Joint Conference, JCKBSE 2014, Volgograd, Russia, September 17-20, 2014. Proceedings 11*, pp. 270–285.

Ranawana, R. & Palade, V. (2006). Optimized precision-a new measure for classifier performance evaluation. *2006 IEEE international conference on evolutionary computation*, pp. 2254–2261.

Rathee, A. & Chhabra, J. K. (2017). Restructuring of object-oriented software through cohesion improvement using frequent usage patterns. *ACM SIGSOFT Software Engineering Notes*, 42(3), 1–8.

Razali, N. M., Wah, Y. B. et al. (2011). Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests. *Journal of statistical modeling and analytics*, 2(1), 21–33.

RDocumentation. [[Accessed 20-Dec-2022]]. (2022a). cohen.d function - RDocumentation — rdocumentation.org. Retrieved from: https://www.rdocumentation.org/packages/effsize/versions/0.8.1/topics/cohen.d.

RDocumentation. [[Accessed 20-Dec-2022]]. (2022b). wilcox.test function - RDocumentation — rdocumentation.org. Retrieved from: https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/wilcox.test.

Rebatchi, H., Bissyandé, T. F. & Moha, N. (2024). Dependabot and security pull requests: large empirical study. *Empirical Software Engineering*, 29(5), 128.

Reiss, R.-D., Thomas, M. & Reiss, R. (1997). *Statistical analysis of extreme values*. Springer.

Ren, K., Yang, H., Zhao, Y., Chen, W., Xue, M., Miao, H., Huang, S. & Liu, J. (2018). A robust AUC maximization framework with simultaneous outlier detection and feature selection for positive-unlabeled classification. *IEEE transactions on neural networks and learning systems*, 30(10), 3072–3083.

Rigby, P. C. & Bird, C. (2013). Convergent contemporary software peer review practices. *Proceedings of the 2013 9th joint meeting on foundations of software engineering*, pp. 202–212.

Rigby, P. C. & Storey, M.-A. (2011). Understanding broadcast based peer review on open source software projects. *Proceedings of the 33rd International conference on software engineering*, pp. 541–550.

Rigby, P. C., German, D. M. & Storey, M.-A. (2008). Open source software peer review practices: a case study of the apache server. *Proceedings of the 30th international conference on Software engineering*, pp. 541–550.

Riley-Bennett, F. & Bennett, S. (2024). Applying the Nominal Group Technique for Structured Consensus Building. *Placeholder — please verify authors, title, venue, and year*. Citation key used in Chapter 9; full metadata pending verification.

Riungu-Kalliosaari, L., Mäkinen, S., Lwakatare, L. E., Tiihonen, J. & Männistö, T. (2016). DevOps adoption benefits and challenges in practice: A case study. *Product-Focused Software Process Improvement: 17th International Conference, PROFES 2016, Trondheim, Norway, November 22-24, 2016, Proceedings 17*, pp. 590–597.

Robles, B. D. V., Lara, I. L. A., Salgado, R. S. & Hidalgo-Reyes, M. (2023). Identification of methods, approaches, and factors in effort estimation for DevOps projects: a systematic literature mapping. *2023 Mexican International Conference on Computer Science (ENC)*, pp. 1–6.

Roche, J. (2013). Adopting DevOps practices in quality assurance. *Communications of the ACM*, 56(11), 38–43.

Rodriguez, D., Ruiz, R., Riquelme, J. C. & Harrison, R. (2013). A study of subgroup discovery approaches for defect prediction. *Information and Software Technology*, 55(10), 1810–1822.

Rodriguez-Galiano, V. F., Ghimire, B., Rogan, J., Chica-Olmo, M. & Rigol-Sanchez, J. P. (2012). An assessment of the effectiveness of a random forest classifier for land-cover classification. *ISPRS journal of photogrammetry and remote sensing*, 67, 93–104.

Romano, J., Kromrey, J. D., Coraggio, J., Skowronek, J. & Devine, L. (2006). Exploring Methods for Evaluating Group Differences on the NSSE and Other Surveys: Are the t-test and Cohen's d Indices the Most Appropriate Choices? *Proceedings of the Annual Meeting of the Southern Association for Institutional Research*, pp. 1–51.

Roque, L., Araújo, A. A., Dantas, A., Saraiva, R. & Souza, J. (2016). Human Resource Allocation in Agile Software Projects Based on Task Similarities. *Search Based Software Engineering*, pp. 291–297. doi: 10.1007/978-3-319-47106-8_25.

Ruk, S. A., Mahmood, Y., Azmi, A., Azmi, N. F. M. & Gul, S. (2025a). Effort Estimation in Agile and DevOps: A Systematic Literature Review of Stakeholder Dissatisfaction. *IEEE Access*.

Ruk, S. A., Mahmood, Y., Azmi, A., Firdaus Mohd Azmi, N. & Gul, S. (2025b). Effort Estimation in Agile and DevOps: A Systematic Literature Review of Stakeholder Dissatisfaction. *IEEE Access*, 13, 167925-167941. doi: 10.1109/ACCESS.2025.3600321.

Rzig, D. E., Hassan, F. & Kessentini, M. (2022). An empirical study on ML DevOps adoption trends, efforts, and benefits analysis. *Information and Software Technology*, 152, 107037.

Saculinggan, M. & Balase, E. A. (2013). Empirical power comparison of goodness of fit tests for normality in the presence of outliers. *Journal of Physics: Conference Series*, 435(1), 012041.

Sadovykh, A., Widforss, G., Truscan, D., Enoiu, E. P., Mallouli, W., Iglesias, R., Bagnto, A. & Hendel, O. (2021). Veridevops: Automated protection and prevention to meet security requirements in devops. *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1330–1333.

Sadowski, C., Van Gogh, J., Jaspan, C., Soderberg, E. & Winter, C. (2015). Tricorder: Building a program analysis ecosystem. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, 1, 598–608.

Sadowski, C., Söderberg, E., Church, L., Sipko, M. & Bacchelli, A. (2018). Modern Code Review: A Case Study at Google. *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 181–190. doi: 10.1145/3183519.3183525.

Sahni, B. Overview on Estimation in Agile Methodology. *J Artif Intell Mach Learn & Data Sci 2022*, 1(1), 511–514.

Saidani, I., Ouni, A., Chouchane, M. & Mkaouer, M. W. (2020). Predicting Continuous Integration Build Failures Using Evolutionary Search. *Information and Software Technology*, 128, 106393. doi: 10.1016/j.infsof.2020.106393.

Saini, N. & Britto, R. (2021). Using machine intelligence to prioritise code review requests. *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 11–20.

Sallin, M., Kropp, M., Anslow, C., Quilty, J. W. & Meier, A. (2021). Measuring software delivery performance using the four key metrics of devops. *Agile Processes in Software Engineering and Extreme Programming: 22nd International Conference on Agile Software Development, XP 2021, Virtual Event, June 14–18, 2021, Proceedings*, pp. 103–119.

Sandeep, R., Sánchez-Gordón, M., Colomo-Palacios, R. & Kristiansen, M. (2022). Effort estimation in agile software development: a exploratory study of practitioners' perspective. *International Conference on Lean and Agile Software Development*, pp. 136–149.

Schroeder, J., Berger, C., Knauss, A., Preenja, H., Ali, M., Staron, M. & Herpel, T. (2017). Comparison of model size predictors in practice. *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pp. 186–188.

Schumann, J., Gundy-Burlet, K., Pasareanu, C., Menzies, T. & Barrett, A. (2009). Software V&V support by parametric analysis of large software simulation systems. *2009 IEEE Aerospace conference*, pp. 1–8.

Seo, S. (2006). *A review and comparison of methods for detecting outliers in univariate data sets*. (Ph.D. thesis, University of Pittsburgh).

Serrador, P. & Pinto, J. K. (2015). Does Agile work?—A quantitative analysis of agile project success. *International journal of project management*, 33(5), 1040–1051.

Shafikuzzaman, M. (2024). *Leveraging Zero-Shot and Few-Shot Learning for Enhanced Sentiment and Requirement Classification in Software Engineering*. (Master's thesis, Lamar University-Beaumont).

Shafiq, S., Mashkoor, A., Mayr-Dorn, C. & Egyed, A. (2021). TaskAllocator: A Recommendation Approach for Role-based Tasks Allocation in Agile Software Development. *2021 IEEE/ACM Joint 15th International Conference on Software and System Processes (ICSSP) and 16th ACM/IEEE International Conference on Global Software Engineering (ICGSE)*, pp. 39–49. doi: 10.1109/ICSSP-ICGSE52873.2021.00014.

Shapiro, S. S., Wilk, M. B. & Chen, H. J. (1968). A comparative study of various tests for normality. *Journal of the American statistical association*, 63(324), 1343–1372.

Sharma, R., Chen, F., Fard, F. & Lo, D. (2022). An exploratory study on code attention in BERT. *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, pp. 437–448.

Sharma, T., Jatain, A., Bhaskar, S. & Pabreja, K. (2023). Ensemble Machine Learning Paradigms in Software Defect Prediction. *Procedia Computer Science*, 218, 199–209.

Sinaga, B. L., Ahmad, S., Abas, Z. A. & Anggarajati, A. W. (2022). The impact of training data selection on the software defect prediction performance and data complexity. *Bulletin of Electrical Engineering and Informatics*, 11(5), 2903–2921.

Sinayobye, J. O., Kaawaase, K. S., Kiwanuka, F. N. & Musabe, R. (2019). Hybrid model of correlation based filter feature selection and machine learning classifiers applied on smart meter data set. *2019 IEEE/ACM Symposium on Software Engineering in Africa (SEiA)*, pp. 1–10.

singh, R. [[Accessed 20-Dec-2022]]. (2022). It's all about Outliers — medium.com. Retrieved from: https://medium.com/analytics-vidhya/its-all-about-outliers-cbe172aa1309.

Śliwerski, J., Zimmermann, T. & Zeller, A. (2005). When do changes induce fixes? *ACM sigsoft software engineering notes*, 30(4), 1–5.

Song, L. & Minku, L. L. (2022). A Procedure to Continuously Evaluate Predictive Performance of Just-In-Time Software Defect Prediction Models During Software Development. *IEEE Transactions on Software Engineering*.

Song, Q., Guo, Y. & Shepperd, M. (2018). A comprehensive investigation of the role of imbalanced learning for software defect prediction. *IEEE Transactions on Software Engineering*, 45(12), 1253–1269.

Spadini, D., Aniche, M., Storey, M.-A., Bruntink, M. & Bacchelli, A. (2018). When testing meets code review: Why and how developers review tests. *Proceedings of the 40th International Conference on Software Engineering*, pp. 677–687.

Spearman, C. (1961). The proof and measurement of association between two things.

Stang, A., Deckert, M., Poole, C. & Rothman, K. J. (2016). Statistical inference in abstracts of major medical and epidemiology journals 1975-2014: a systematic review. *European journal of epidemiology*.

Stanley, C. & Byrne, M. D. (2013). Predicting tags for stackoverflow posts. *Proceedings of ICCM*, 2013.

Staron, M., Hansson, J., Feldt, R., Henriksson, A., Meding, W., Nilsson, S. & Höglund, C. (2013). Measuring and visualizing code stability–a case study at three companies. *2013 Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement*, pp. 191–200.

Svyatkovskiy, A., Deng, S. K., Fu, S. & Sundaresan, N. (2020). Intellicode compose: Code generation using transformer. *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1433–1443.

Taibi, D., Lenarduzzi, V. & Pahl, C. (2019). Continuous architecting with microservices and devops: A systematic mapping study. *Cloud Computing and Services Science: 8th International Conference, CLOSER 2018, Funchal, Madeira, Portugal, March 19-21, 2018, Revised Selected Papers 8*, pp. 126–151.

Tantithamthavorn, C. & Hassan, A. E. (2018). An experience report on defect modelling in practice: Pitfalls and challenges. *Proceedings of the 40th International conference on software engineering: Software engineering in practice*, pp. 286–295.

Tantithamthavorn, C., McIntosh, S., Hassan, A. E., Ihara, A. & Matsumoto, K. (2015). The impact of mislabelling on the performance and interpretation of defect prediction models. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, 1, 812–823.

Tantithamthavorn, C., McIntosh, S., Hassan, A. E. & Matsumoto, K. (2016a). Automated parameter optimization of classification techniques for defect prediction models. *Proceedings of the 38th international conference on software engineering*, pp. 321–332.

Tantithamthavorn, C., McIntosh, S., Hassan, A. E. & Matsumoto, K. (2016b). An empirical comparison of model validation techniques for defect prediction models. *IEEE Transactions on Software Engineering*, 43(1), 1–18.

Tantithamthavorn, C., Hassan, A. E. & Matsumoto, K. (2018). The impact of class rebalancing techniques on the performance and interpretation of defect prediction models. *IEEE Transactions on Software Engineering*, 46(11), 1200–1219.

Thongtanunam, P., McIntosh, S., Hassan, A. E. & Iida, H. (2015). Investigating code review practices in defective files: An empirical study of the qt system. *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pp. 168–179.

Thongtanunam, P., McIntosh, S., Hassan, A. E. & Iida, H. (2016a). Revisiting code ownership and its relationship with software quality in the scope of modern code review. *Proceedings of the 38th international conference on software engineering*, pp. 1039–1050.

Thongtanunam, P., Tantithamthavorn, C., McIntosh, S., Hassan, A. E. & Iida, H. (2016b). Who Should Review My Code? A File Location-Based Code-Reviewer Recommendation Approach. *SANER*, pp. 141–150. doi: 10.1109/SANER.2016.21.

Thongtanunam, P., McIntosh, S., Hassan, A. E. & Iida, H. (2017a). Review Participation in Modern Code Review. *Empirical Software Engineering*, 22(2), 768–817. doi: 10.1007/s10664-016-9452-6.

Thongtanunam, P., McIntosh, S., Hassan, A. E. & Iida, H. (2017b). Review participation in modern code review: An empirical study of the android, Qt, and OpenStack projects. *Empirical Software Engineering*, 22, 768–817.

Thukkaram, M. (2025). Estimating Agile Effort through Merge Request Analytics with an Explainable T-Shirt Sizing Model.

Thurmond, V. A. (2001). The point of triangulation. *Journal of nursing scholarship*, 33(3), 253–258.

Tian, Y., Lawall, J. & Lo, D. (2012). Identifying linux bug fixing patches. *2012 34th international conference on software engineering (ICSE)*, pp. 386–396.

Tian, Y., Nagappan, M., Lo, D. & Hassan, A. E. (2015). What are the characteristics of high-rated apps? a case study on free android applications. *2015 IEEE international conference on software maintenance and evolution (ICSME)*, pp. 301–310.

Tmap. [[Accessed 02-Apr-2023]]. (2023). Anomaly management | DevOps — tmap.net. Retrieved from: https://www.tmap.net/building-blocks/Anomaly-management-DevOps.

Tomek, I. (1976). Two modifications of CNN.

Torres, A., Galante, R., Pimenta, M. S. & Martins, A. J. B. (2017). Twenty years of object-relational mapping: A survey on patterns, solutions, and their implications on application design. *Information and Software Technology*, 82, 1–18. doi: 10.1016/j.infsof.2016.09.009.

Trudel, S. & Boisvert, M. (2011). *Choisir l'agilité*. [S.l.]: Dunod. Retrieved from: https://research.ebsco.com/linkprocessor/plink?id=826e63e4-6107-36b4-8909-ec91fba52433.

TrustRadius. (2022). *Current Trends in Code Review 2022*. Retrieved from: https://media.trustradius.com/product-downloadables/DD/D7/XID8MVZTH0JF.pdf.

Tsai, H.-T., Moskowitz, H. & Lee, L.-H. (2003). Human resource selection for software development projects using Taguchi's parameter design. *European Journal of Operational Research*, 151(1), 167–180. doi: 10.1016/S0377-2217(02)00600-8.

Tsay, J., Dabbish, L. & Herbsleb, J. (2014a). Let's Talk About It: Evaluating Contributions Through Discussion in GitHub. *FSE*, pp. 144–154. doi: 10.1145/2635868.2635882.

Tsay, J., Dabbish, L. & Herbsleb, J. (2014b). Let's talk about it: evaluating contributions through discussion in GitHub. *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*, pp. 144–154.

Tufano, M., Pantiuchina, J., Watson, C., Bavota, G. & Poshyvanyk, D. (2019a). On learning meaningful code changes via neural machine translation. *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pp. 25–36.

Tufano, M., Watson, C., Bavota, G., Penta, M. D., White, M. & Poshyvanyk, D. (2019b). An empirical study on learning bug-fixing patches in the wild via neural machine translation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 28(4), 1–29.

Tufano, R., Pascarella, L., Tufano, M., Poshyvanyk, D. & Bavota, G. (2021). Towards automating code review activities. *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 163–174.

Tufano, R., Masiero, S., Mastropaolo, A., Pascarella, L., Poshyvanyk, D. & Bavota, G. (2022). Using pre-trained models to boost code review automation. *Proceedings of the 44th international conference on software engineering*, pp. 2291–2302.

Tunkel, S. & Herbold, S. (2022). Exploring the relationship between performance metrics and cost saving potential of defect prediction models. *Empirical Software Engineering*, 27(7), 1–38.

Tunstall, L., Reimers, N., Jo, U. E. S., Bates, L., Korat, D., Wasserblat, M. & Pereg, O. (2022). Efficient few-shot learning without prompts. *arXiv preprint arXiv:2209.11055*.

Turhan, B., Menzies, T., Bener, A. B. & Di Stefano, J. (2009). On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering*, 14(5), 540–578.

Turzo, A. K. & Bosu, A. (2024). What makes a code review useful to opendev developers? an empirical investigation. *Empirical Software Engineering*, 29(1), 6.

Tyagi, S. & Mittal, S. (2020). Sampling approaches for imbalanced data classification problem in machine learning. *Proceedings of ICRIC 2019: Recent Innovations in Computing*, pp. 209–221.

Usman, M., Britto, R., Damm, L.-O. & Börstler, J. (2018). Effort estimation in large-scale software development: An industrial case study. *Information and Software Technology*, 99, 21–40. doi: 10.1016/j.infsof.2018.02.009.

Vacanti, D. & Coleman, J. (2020). Kanban Guide. Retrieved on 2025-01-24 from: https://kanbanguides.org/english/.

Van Hulse, J., Khoshgoftaar, T. M. & Napolitano, A. (2009). An empirical comparison of repetitive undersampling techniques. *2009 IEEE international conference on information reuse & integration*, pp. 29–34.

Vanderster, D. C., Dimopoulos, N. J., Parra-Hernandez, R. & Sobie, R. J. (2009). Resource allocation on computational grids using a utility model and the knapsack problem. *Future Generation Computer Systems*, 25(1), 35–50. doi: 10.1016/j.future.2008.07.006.

Vassallo, C., Proksch, S., Gall, H. C. & Di Penta, M. (2019). Automated reporting of anti-patterns and decay in continuous integration. *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pp. 105–115.

Village, E. [[Accessed 11-Dec-2022]]. (2022). Engineering Village - Error — engineeringvillage.com. Retrieved from: https://www.engineeringvillage.com/search/quick.url.

Wahono, R. S. (2015). A systematic literature review of software defect prediction. *Journal of software engineering*, 1(1), 1–16.

Wang, H. & Song, M. (2011). Ckmeans. 1d. dp: optimal k-means clustering in one dimension by dynamic programming. *The R journal*, 3(2), 29.

Wang, H., Khoshgoftaar, T. M. & Napolitano, A. (2010). A comparative study of ensemble feature selection techniques for software defect prediction. *2010 Ninth International Conference on Machine Learning and Applications*, pp. 135–140.

Wang, H., Khoshgoftaar, T. M. & Napolitano, A. (2014). Choosing the Best Classification Performance Metric for Wrapper-based Software Metric Selection for Defect Prediction. *SEKE*, pp. 540–545.

Wang, S., Chen, T.-H. & Hassan, A. E. (2018). Understanding the factors for fast answers in technical Q&A websites. *Empirical Software Engineering*, 23(3), 1552–1593.

Wang, S., Huang, L., Gao, A., Ge, J., Zhang, T., Feng, H., Satyarth, I., Li, M., Zhang, H. & Ng, V. (2022). Machine/Deep Learning for Software Engineering: A Systematic Literature Review. *IEEE Transactions on Software Engineering*.

Wang, S., Bansal, C., Nagappan, N. & Philip, A. A. (2019). Leveraging change intents for characterizing and identifying large-review-effort changes. *Proceedings of the Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering*, pp. 46–55.

Wang, S., Bansal, C. & Nagappan, N. (2021). Large-scale intent analysis for identifying large-review-effort code changes. *Information and Software Technology*, 130, 106408.

Wang, X., Conboy, K. & Cawley, O. (2012). "Leagile" software development: An experience report analysis of the application of lean approaches in agile software development. *Journal of Systems and Software*, 85(6), 1287–1299. doi: 10.1016/j.jss.2012.01.061.

Wang, X. & Wu, K. (2022). Research and practice of ministry-level software process improvement based on SW- CMM model. *Proceedings of the 2022 5th International Conference on Software Engineering and Information Management*, pp. 61–67.

Waske, B., Heinzel, V., Braun, M. & Menz, G. (2007). Random forests for classifying multi-temporal sar data. *Proc.'Envisat Symposium*, 2007, 23–27.

Watts, J. & Lawrence, R. (2008). Merging random forest classification with an object-oriented approach for analysis of agricultural lands. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 37(B7), 2008.

Weflen, E., MacKenzie, C. A. & Rivero, I. V. (2022). An influence diagram approach to automating lead time estimation in Agile Kanban project management. *Expert Systems with Applications*, 187, 115866.

Weigelt, K. E. Challenges and Mitigation Propositions for Effort Estimation in Large-Scale Agile Development.

White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J. & Schmidt, D. C. (2023). A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. *arXiv preprint arXiv:2302.11382*.

Whitehead, J. (2007). Collaboration in software engineering: A roadmap. *Future of Software Engineering (FOSE'07)*, pp. 214–225.

William, P., Kumar, P., Chhabra, G. S. & Vengatesan, K. (2021a). Task Allocation in Distributed Agile Software Development using Machine Learning Approach. *2021 International Conference on Disruptive Technologies for Multi-Disciplinary Research and Applications (CENTCON)*, 1, 168–172. doi: 10.1109/CENTCON52345.2021.9688114.

William, P., Kumar, P., Chhabra, G. & Vengatesan, K. (2021b). Task allocation in distributed agile software development using machine learning approach. *2021 International conference on disruptive technologies for multi-disciplinary research and applications (CENTCON)*, 1, 168–172.

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, pp. 1–10.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., Wesslén, A. et al. (2012). *Experimentation in software engineering*. Springer.

Xia, W., Wen, Y., Foh, C. H., Niyato, D. & Xie, H. (2014a). A survey on software-defined networking. *IEEE Communications Surveys & Tutorials*, 17(1), 27–51.

Xia, X., Lo, D., Qiu, W., Wang, X. & Zhou, B. (2014b). Automated configuration bug report prediction using text mining. *2014 IEEE 38th annual computer software and applications conference*, pp. 107–116.

Xia, Y., Yan, G. & Zhang, H. (2014c). Analyzing the significance of process metrics for TT&C software defect prediction. *2014 IEEE 5th International Conference on Software Engineering and Service Science*, pp. 77–81.

Xu, B., Lo, D., Xia, X., Sureka, A. & Li, S. (2015). Efspredictor: Predicting configuration bugs with ensemble feature selection. *2015 Asia-Pacific Software Engineering Conference (APSEC)*, pp. 206–213.

Xu, Z., Pang, S., Zhang, T., Luo, X.-P., Liu, J., Tang, Y.-T., Yu, X. & Xue, L. (2019). Cross project defect prediction via balanced distribution adaptation based transfer learning. *Journal of Computer Science and Technology*, 34(5), 1039–1062.

Yang, L., Xu, J., Zhang, Y., Zhang, H. & Bacchelli, A. (2023). EvaCRC: Evaluating Code Review Comments. *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 275–287.

Yang, L., Liu, B., Jia, J., Xu, J., Xue, J., Zhang, H. & Bacchelli, A. (2025). Prioritizing code review requests to improve review efficiency: a simulation study. *Empirical Software Engineering*, 30(1), 24.

Yi, M., Fan, G., Yu, H. & Yang, X. (2026). An Empirical Study on the Impact of Class Overlap in Just-in-Time Software Defect Prediction.

Yin, Y., Zhao, Y., Sun, Y. & Chen, C. (2023). Automatic code review by learning the structure information of code graph. *Sensors*, 23(5), 2551.

Yin, Z., Ma, X., Zheng, J., Zhou, Y., Bairavasundaram, L. N. & Pasupathy, S. (2011). An empirical study on configuration errors in commercial and open source systems. *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pp. 159–172.

Yu, X., Liu, J., Yang, Z., Jia, X., Ling, Q. & Ye, S. (2017a). Learning from imbalanced data for predicting the number of software defects. *2017 IEEE 28th international symposium on software reliability engineering (ISSRE)*, pp. 78–89.

Yu, X., Wu, M., Zhang, Y. & Fu, M. (2017b). Combing Data Filter and Data Sampling for Cross-Company Defect Prediction: An Empricial Study. *SEKE*, pp. 301–306.

Yu, Y., Wang, H., Filkov, V., Devanbu, P. & Vasilescu, B. (2015). Wait for it: Determinants of pull request evaluation latency on github. *2015 IEEE/ACM 12th working conference on mining software repositories*, pp. 367–371.

Zachary, C. L. (2016). The mythos of model interpretability. *Communications of the ACM*, 1–6.

Zampetti, F., Bavota, G., Canfora, G. & Di Penta, M. (2019). A Study on the Interplay between Pull Request Review and Continuous Integration Builds. *SANER*, pp. 38–48. doi: 10.1109/SANER.2019.8667996.

Zanjani, M. B., Kagdi, H. & Bird, C. (2016). Automatically Recommending Peer Reviewers in Modern Code Review. *IEEE TSE*, 42(6), 530–543. doi: 10.1109/TSE.2015.2500238.

Zhang, F., Khomh, F., Zou, Y. & Hassan, A. E. (2012). An empirical study on factors impacting bug fixing time. *2012 19th Working conference on reverse engineering*, pp. 225–234.

Zhang, L., Cao, C., Wang, Z. & Liu, P. (2023a). Which Features are Learned by CodeBert: An Empirical Study of the BERT-based Source Code Representation Learning. *arXiv preprint arXiv:2301.08427*.

Zhang, X., Yu, Y., Gousios, G. & Rastogi, A. (2022). Pull request decisions explained: An empirical overview. *IEEE Transactions on Software Engineering*, 49(2), 849–871.

Zhang, X., Yu, Y., Gousios, G. & Rastogi, A. (2023b). Pull Request Decisions Explained: An Empirical Overview. *IEEE TSE*, 49(2), 849–868. doi: 10.1109/TSE.2022.3146155.

Zhang, Y., He, H., Legunsen, O., Li, S., Dong, W. & Xu, T. (2021). An evolutionary study of configuration design and implementation in cloud systems. *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 188–200.

Zhao, Y., Li, L., Wang, H., Cai, H., Bissyandé, T. F., Klein, J. & Grundy, J. (2021). On the impact of sample duplication in machine-learning-based android malware detection. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 30(3), 1–38.

Zheng, W., Shen, T. & Chen, X. (2021). Just-in-time defect prediction technology based on interpretability technology. *2021 8th International Conference on Dependable Systems and Their Applications (DSA)*, pp. 78–89.

Zimmermann, T. & Nagappan, N. (2008). Predicting defects using network analysis on dependency graphs. *Proceedings of the 30th international conference on Software engineering*, pp. 531–540.

Zimmermann, T., Premraj, R. & Zeller, A. (2007). Predicting defects for eclipse. *Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007)*, pp. 9–9.