Leveraging Merge Request Data to Analyze DevOps Practices: Insights from a Networking Software Solution Company

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

[1]École de technologie supérieure (ÉTS), Montréal, Canada
[2]TELUS, Toronto, Canada

Abstract. GitLab's Merge Request (MR) mechanism is a cornerstone of DevOps, traditionally used for code review and analysis. This study broadens its scope by leveraging MR data to explore multiple facets of the DevOps workflow. Using a dataset of 26.7k MRs from 116 projects across four teams in a networking software company, we first examine the impact of environmental and process changes. We analyze how external factors, such as the COVID-19 pandemic, and internal adaptations, like the migration to OpenShift, influence effort, productivity, and collaboration. Our findings indicate that while review effort increased during the pandemic, productivity remained stable, with up to 70% of weekly activities occurring outside standard working hours. Similarly, the OpenShift migration initially disrupted workflows then showed a successful adaptation, with stabilized performance metrics over time. Next, we analyze branch management practices, revealing that stable branches, particularly those linked to new releases, are prioritized, leading to faster review completion. Finally, we apply machine learning (ML) to explain the time to complete code reviews, highlighting the roles of bots and human reviewers in industrial context. While bots accelerate review initiation, human reviewers play a crucial role in reducing time to complete code review. Additional factors, such as the number of commits and reviewer experience, also significantly influence review efficiency. Our analysis contributes to extending the use of MR data beyond code review by demonstrating how it provides deeper insights into software development workflows, team collaboration, and process adaptations, offering a framework for leveraging MR dynamics to optimize DevOps practices.

Keywords: Software process, DevOps, Merge request, GitLab, Code review

1 Introduction

Research Background: DevOps represents an integrated approach that emphasizes collaboration, automation, and continuous improvement throughout the software development lifecycle [9, 23]. By integrating DevOps practices, organizations can significantly enhance their agility, shorten the time to market, increase automation, and ensure more reliable and consistent software releases [4]. DevOps integrates the efforts of development and operations teams, facilitates continuous integration, continuous delivery, and continuous deployment, and fosters a culture of collaboration and shared responsibility to address common bottlenecks and inefficiencies in traditional software development practices [11]. The DevOps process is structured into distinct phases, each crucial for maintaining a smooth and efficient workflow, which are: development, testing, building and monitoring. Each phase necessitates specific checks and validations before progressing to the next phase. By leveraging DevOps within the framework of SPI, organizations can achieve a streamlined and effective software development lifecycle, ultimately leading to higher productivity, better software quality, and reduced development costs and time [20, 22, 24].

To realize the DevOps process, multiple tools and technologies are employed at each stage, ensuring seamless execution and continuous improvement throughout the software development process. For instance, in the development stage, version control systems like Git are used, while in the testing stage, continuous integration (CI) servers such as GitHub

David C. Wyld et al. (Eds): NLCAI, AIFU, CCSEA, BIoT, SEA, SIPRO, BDML, CLOUD – 2025
pp. 219-238, 2025. CS & IT - CSCP 2025
DOI: 10.5121/csit.2025.151018

Actions or GitLab CI are common. Additionally, there are different mechanisms that help streamline the workflow, such as GitLab's merge request and Git hooks.

Research Motivation: GitLab's Merge Request (MR) mechanism serves as a central element within the DevOps pipeline, streamlining the submission, review, and integration of new code changes into the main code base (from source to target branch) [1]. Throughout this process, collaborators actively participate in discussions, offering insights and improvements to the code under review. In addition, the MR encapsulates various aspects of the project and team dynamics. Analyzing this data provides a rich perspective on multiple dimensions of the software system. It facilitates productivity analysis by quantifying effort and contributions, identifies workflow bottlenecks to optimize code updates, ensures quality of work through thorough reviews, and improves collaboration patterns by identifying key contributors. It also provides insight into how the development process is impacted by various factors, such as the impact of new technology integration on release velocity.

Research Objectives: While MR data have been used in the literature to analyze different aspects of the code review process [8, 5, 18, 10], the goal of this paper is to examine various aspects of DevOps processes using GitLab MR data to identify and improve areas affected by different changes in the process. To do so, we analyze 26,7k MRs data from four teams of our industrial partner, a networking software solution company that deploys 5G and cloud edge fabric technologies for telco, data center, and cloud service provider customers using a DevOps approach for efficient software development and deployment. These main aspects are addressed in this study:

(1) Impact of Environmental or Process Changes: Changes within or outside the company can significantly affect team dynamics and performance. Analyzing MR data reveals how these changes impact productivity, collaboration, and effort, helping us address two key questions:

a- Environmental Changes: The COVID-19 pandemic's onset in Canada on March 16, 2020, forced an immediate shift from office to remote work, disrupting daily interactions around development and code reviews. This study analyzes MR data to examine the pandemic's impact on developer activities and how such external disruptions influence the DevOps process and organizational adaptability, answering the research question: ***How have environmental changes, specifically the shift from office-based work to remote work during the COVID-19 pandemic, impacted workplace dynamics and productivity?*** Analyzing MR data shows increased effort and longer times to complete code review during COVID-19, with more activity outside regular hours and weekends, which persist after the pandemic, indicating greater flexibility. Despite these changes, productivity remained stable, demonstrating team adaptability.

b- Process Changes:Process changes, like integrating new technologies, can disrupt DevOps workflows, requiring developers to learn new skills and manage increased bugs. A key example is the company's migration to OpenShift (May 1, 2020 – Jan 31, 2021), where OpenShift-related MRs were separated to ensure a smooth transition. This study compares OpenShift and non-OpenShift MRs to assess the migration's impact on the overall process. Our research question is: ***Has there been any impact of the process changes, specifically the migration to OpenShift technology, on different aspect of the organization?*** During the migration, we observe unstable times to complete code review and smaller MRs on OpenShift due to gradual technology integration and code changes. By the final sprints of the migration, times to complete code review and MR sizes stabilized, showing increased developer competency with OpenShift.

[1] `https://docs.gitlab.com/ee/user/project/merge_requests/`

(2) Branch Information Analysis: Branch policies are crucial for efficient DevOps, enabling teams to work independently without disrupting the main codebase. Analyzing branch usage helps track progress, detect late-stage bugs, and reveal team workflows, such as release frequency. MR data, which records source and target branches, offers valuable insights into branch management. This study explores different branch types and their management using MR data, answering: ***How are different types of GitLab branches used and managed in an industrial DevOps environment?*** We found that MRs on stable branches are prioritized, completed faster, and receive quicker responses, likely due to their connection to release production and bug management.

(3) Code Review Process Analysis: Time to complete code review is crucial for understanding the review process and is widely regarded as a key metric for estimating MR effort. Zhang et al. [30] found that faster first comments accelerate pull request closures but did not distinguish between bots and humans, while Hasan et al. [8] suggested bot responses weaken this effect. Since automation is central to DevOps, we leverage Machine Learning (ML) to explain the time to complete code review, focusing on the impact of human vs. automated bot first comments. Our research question is: ***To what extent does the origin of the first comment—bot or human—impact the time to complete code reviews?*** We found that bot-generated first comments dominate initial MR interactions, significantly speeding up the initiation of the review process, with median response times ranging from 4.46 to 23.78 minutes. However, it is the human comments that have the most significant impact on reducing time to complete code review, explaining over 93% of the variance in MR completion time. This finding highlights the critical role of timely human feedback in driving efficient code reviews, while also acknowledging the importance of bots in initiating the process. Other factors, such as the number of commits and the experience of the reviewers, were also found to play a significant role in determining the overall efficiency of the code review process.

Research Contribution: The main contribution of this study lies in highlighting the broader significance of MR data, extending its analysis beyond traditional code review to provide insights into various aspects of the DevOps process. By examining MR data, we gain a deeper understanding of not only the quality of code reviews but also how teams collaborate and how external factors, such as technological changes or unexpected events, influence the development workflow. Additionally, this study offers a valuable analysis of the factors impacting review completion time and the timing of the first comment in an industrial context. Our research provides practical guidance for practitioners on leveraging code review data, particularly MR data, to enhance their DevOps processes through more in-depth analysis and data-driven decision-making. The paper proceeds with related work in Section 2, followed by the methodology in Section 3. Section 4 addresses the research questions, concluding with discussions on threats to validity in Section 5 and the paper's conclusion in Section 6.

2 Background and Related Work

MR Mechanisms in DevOps: MR mechanism plays a central role in the DevOps process, structuring the proposal, review, and integration of code changes. Throughout the MR lifecycle, contributors engage in discussions, revisions, and quality assurance before the code is merged. This structured process enforces quality control while streamlining development workflows. Similar mechanisms exist on other platforms. GitHub's pull request (PR) system and Gerrit's code review framework offer comparable functionality, though with variations in review policies and approval processes. As discussed by Tufano et al.

[29], automated tools supporting review activities play a crucial role in accelerating the review process while maintaining software quality.

Code Review Practices and Efficiency: Numerous studies have sought to analyze and improve the efficiency of the code review process. For instance, Chouchen et al. [5] and Maddila et al. [18] developed predictive models for estimating review duration based on factors such as code complexity, reviewer expertise, and discussion volume. Hasan et al. [8] found that shorter time-to-universal-first-response correlates with quicker pull request completion, though bot-first responses were less impactful than human interventions. Similarly, Bosu and Carver [3] showed that established developers receive faster feedback, leading to shorter review times. The number and experience of reviewers also influence review dynamics. Jiang, Adams, and German [12] demonstrated that increasing the number of reviewers extends the review process, while Thongtanunam et al. [28] explored how specific code patch characteristics attract more reviewers. Additionally, Baysal et al. [2] highlighted that non-technical factors, such as team hierarchy and social relationships, impact review duration. ML techniques have been widely applied to predict review outcomes and optimize review processes. Fan et al. [6] and Islam et al. [10] used ML models to predict whether a code change would be merged or abandoned. Li et al. [17] analyzed redundant pull requests, showing that duplicate contributions increase inefficiencies. Additionally, Khatoonabadi et al. [13] investigated the abandonment of pull requests, identifying inexperience, high complexity, and prolonged review cycles as key contributors. Finally, Golzadeh et al. [7] developed a classification model for detecting bot-generated comments on GitHub, highlighting their benefits in expediting reviews but also their limitations in nuanced decision-making.

Code Evolution Through Code Review Data: Several studies have also analyzed development aspects using the data provided from the code review process. For instance, Nejati, Alfadel, and McIntosh [21] found that build specification changes in projects like Qt and Eclipse receive less attention during reviews than production and test code, despite posing a higher defect risk. Similarly, Spadini et al. [25] observed that production code undergoes more thorough reviews than test code, impacting overall software stability. To investigate code defects, Thongtanunam et al. [27] demonstrated that defective files often receive less rigorous reviews, potentially leading to overlooked issues. Additionally, Staron et al. [26] showed that modifications to existing code can significantly impact software stability, influencing product reliability.

Branches in Code Review: Researchers used the branch information as a metric to study the code review. For example, Lal and Pahwa [15] analysed the code review of software systems using the branches as a metric providing information about code review category or features. AlOmar [1] investigated how refactoring changes are handled in code review by analyzing the dynamics of refactoring branches in the Qt project, examining their impact on review time, feedback patterns, and integration efficiency. Mukadam, Bird, and Rigby [19] examined which software changes are reviewed, who reviews them, how long the review process takes, and the nature of discussions and feedback, using branch information as metric.

Contributions and Research Gaps: Although past studies have explored code review processes and leveraged review data to analyze various aspects—such as defect detection and quality assurance—they, remain primarily focused on the code verification phase rather than examining how review data can be utilized to understand broader DevOps processes. Our study builds on these findings by demonstrating that MR data extends beyond code verification, offering strategic value in tracking development trends in response to environmental and process changes.We also analyze branch management, highlighting

its crucial role in structuring development workflows, prioritizing stable releases, and accelerating review efficiency. Understanding how branches are used in an industrial context allows us to assess their impact on software stability and team collaboration. Finally, automation is increasingly shaping DevOps, with bots assisting in enforcing coding standards and providing initial feedback. We examine the role of automated comments in the time to complete code review, comparing their effectiveness to human comments, following the same work done by Hasan et al. [8] but in industrial context.

3 Methodology

The goal of this section is to discuss the steps we followed to collect and analyze MR data to investigate different aspects of DevOps processes. Our methodology is structured around four RQs, each addressing a distinct dimension of the DevOps workflow through MR analysis. Below, we detail the data collection and metric extraction, keeping details to the approach of each RQ.

3.1 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?We analyze MR activity trends, collaboration patterns, and review effort before, during, and after the transition to remote work to assess its impact on software development productivity.
RQ2: Has there been any impact of the process changes, specifically the migration to OpenShift technology, on different aspects of the organization? We investigate how this technological transition affected MR volume, review efficiency, and developer workflows.
RQ3:How are different types of GitLab branches used and managed in an industrial DevOps environment? We examine branch management practices, focusing on the prioritization of stable branches, review speed, and how different branch types impact the software development lifecycle.
RQ4:To what extent does the origin of the first comment—bot or human—impact the time to complete code reviews? We assess the influence of automated and human-generated comments on review efficiency, exploring whether early feedback accelerates the time to complete the code review and how bots contribute to or hinder the review process.

Table 1: Project teams of our industrial partner

Group	Description	#MR	#Projects
Management Plane (MP)	Manage the communication with Fabric	6,344k	20
Control Plane (CP)	Orchestrate paths for packets and frames	8,396k	31
Data Plane (DP)	Forward packet and frames between interfaces	7,416k	16
FPGA	designs reprogrammable integrated circuits	735	19
Platform (PF)	Manage low-level platforms	4,004k	30

3.2 Data Collection

In this step, we leverage a GitLab extraction tool, initially developed by Legault [16], as the foundation for our custom MR data extraction tool, which utilizes the GitLab API[2]. This tool enables us to systematically collect MR data from four teams working on

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

multiple projects that are described in Table 1. We collected data spanning January 1, 2019, to June 30, 2023, encompassing a wide range of MR attributes, including MR ID, creation date, closure date, associated commits, discussions, notes, and file modifications. The data was retrieved using relevant GitLab API endpoints[3] [4] [5] [6]. To ensure transparency and reproducibility, the MR extraction tool is publicly available[7]. This comprehensive data collection framework provides the necessary flexibility for in-depth analysis, allowing us to examine multiple dimensions of DevOps workflows.

Table 2: List of metrics used in this study

Metric	Description
Creation Week ‡	The week during which the MR was created.
Creation Sprint †	The sprint during which the MR was created.
End Week ‡	The week during which the MR was closed.
Time to complete code review (Lead Time) ‡ † ∗ ⋆	The duration between the creation and closure of the MR.
Creation Hour ‡	The hour when the MR was initially opened.
Closure Hour ‡	The hour when the MR was closed.
Is ORH ‡	A flag that indicates whether the MR includes activities occurring outside of regular working hours.
MR Size † ∗	The amount of lines of code added + deleted related to the MR.
Additions †	The amount of added lines of code related to the MR.
Deletions †	The amount of deleted lines of code related to the MR.
Source Branch ∗	The branch to which the code is added during the MR process.
Target Branch ∗	The branch to which the code is merged upon the closure of the MR.
Time to First Comment ∗ ⋆	Time taken to receive the first comment from the creation of the MR.
#Notes ∗	Number of discussion messages made during the review of the MR.
Description Length ⋆	The length of the description provided for the MR.
Is First MR ⋆	Indicates whether this is the first MR submitted for a given project.
Is Hash-tag ⋆	Indicates whether the "#" tag is present in the MR's title or description.
Is At-Tag Presence ⋆	Indicates whether the "@" tag is present in the MR's title or description.
#Commits ⋆	The number of commits made at the time the MR was opened.
Project Age ⋆	The number of months from the project's creation to the time of the MR submission.
#Opened MRs ⋆	The number of MRs that are open for a given project at the time of analysis.
#Merges ⋆	The number of prior merges in the project.
#Previous MRs ⋆	The number of all previous MRs that have been previously created for the project.
Author as Reviewer ⋆	Indicates whether the MR author is also the reviewer.

‡ **Corresponds to RQ1,** † **Corresponds to RQ2,** ∗ **Corresponds to RQ3,** ⋆ **Corresponds to RQ4**

3.3 Metrics Extraction

Once the raw data is collected, we proceed to calculate various metrics crucial for analyzing different aspects of the DevOps process using MR data. For example, the time to complete code review, which is not directly available from the GitLab API, needs to be calculated as the time between the creation and closure of the MR. In total, 24 metrics, as shown in Table 2, are collected across three dimensions: the MR dimension, which includes metrics related to each MR; the project dimension, which consists of project information; and the developer dimension, which includes metrics related to team members. These metrics provide a comprehensive view of the DevOps process, facilitating detailed analysis and insights.

3.4 Metrics Analysis

We conduct different types of analysis for each RQ. To do this, we filter the data by two time dimensions: by week or by sprint, with a sprint representing the company's three-week development cycle. Once the data is filtered, we use mathematical values such

[3] `https://docs.gitlab.com/api/merge_requests/`
[4] `https://docs.gitlab.com/api/commits/`
[5] `https://docs.gitlab.com/api/discussions/`
[6] `https://docs.gitlab.com/api/notes/`
[7] `https://gitlab.com/ets-devops/merge-requests/mr-analysis-tool`

as mean, median, minimum, and maximum to draw conclusions and compare different aspects. To ensure our analysis is coherent, we fixed several aspects to be analyzed, such as productivity (measured by the number of MRs created and closed each week) and time to complete code review (to measure effort). Each aspect and method is detailed in the approach of each RQ.

4 Results

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 code review on the 22 weeks, which represents the time it takes to complete a code review. This involved analyzing the time taken to complete code reviews both before and after the pandemic, as well as the number of successfully merged MRs. A longer time to complete code 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 code review process, resulting in a slower overall timeline. Conversely, shorter review times 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 [5, 18, 2, 12], we consider the time to complete code 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 pandamic.

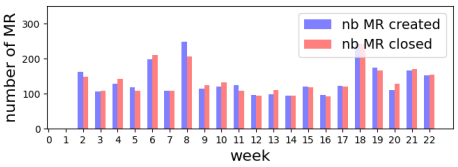

Fig. 1: Number of MR created and ended per week from week 1 to 22

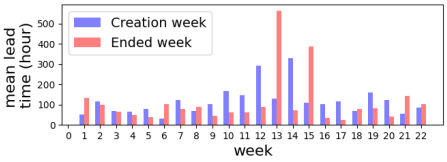

Fig. 2: Meantime to complete code review by week for MR opening and closing from week 1 to 22

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 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, such as weeks 4, 9, and 10, indicating that this is not solely the impact of Covid. 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 code 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 2, the mean time to complete code 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 code review of MRs created in weeks 11 and 12. Regarding the average time to complete code 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.

As shown in table 3, 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 3, there was a notable variation in the percentage of activities done before and after Covid-19. Table 3 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. This pattern clearly illustrates the impact of Covid-19 on developers' temporal autonomy.

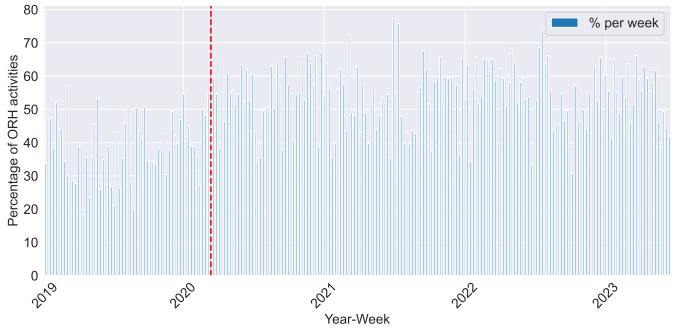

Fig. 3: The distribution of the percentage of out of regular working hours activities by week before and after Covid-19

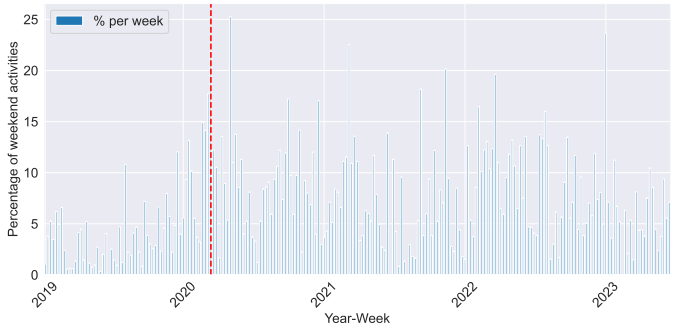

Fig. 4: The distribution of the percentage of weekend activities by week before and after Covid-19

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 3: Out of Regular Hours (ORH) and weekend statistics activities before and after COVID-19

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

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 (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 code 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).

Results

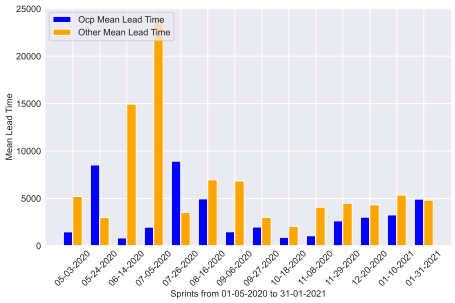

Fig. 5: Mean time to complete code review by sprint for OpenShift and other MRs in the migration period

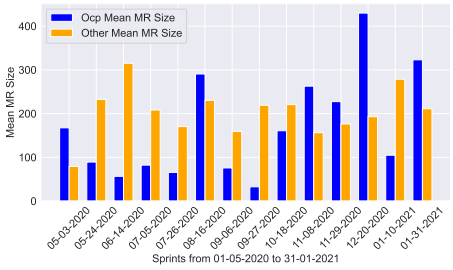

Fig. 6: Mean size by sprint for OpenShift and other MRs in the migration period

Our results show an inverse relationship between the time to complete code review of MRs for OpenShift and other MRs during the first eight sprints, as shown in Figure 5. When significant effort was spent on OpenShift integration, the time to complete code 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 code 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 code 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 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.

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

(a) Openshift MRs (b) Other MRs

Fig. 7: Comparison of additions and deletions per MR in Ocp and other 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 code 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 [26].

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 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 (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.

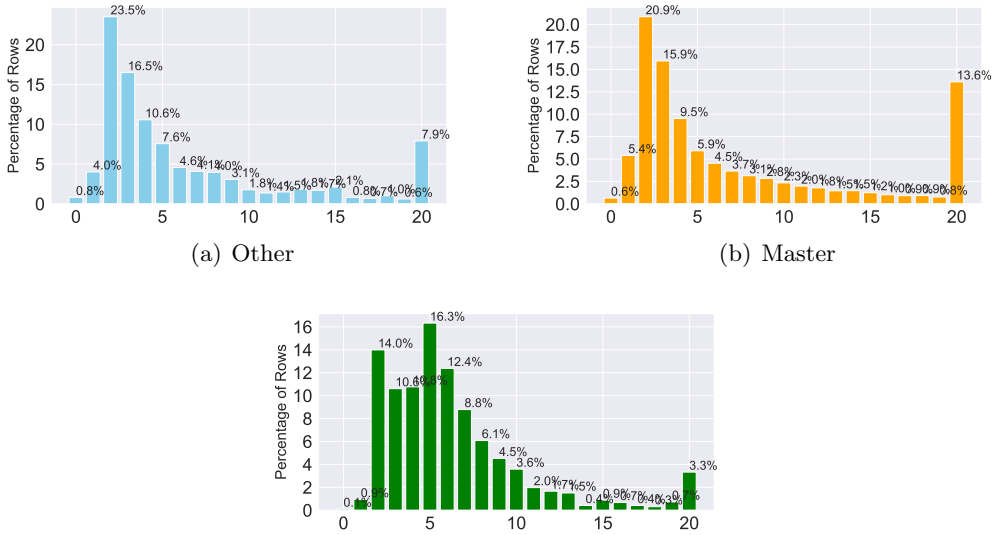

Fig. 8: Distribution of #notes by branch

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 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 10 demonstrates that MRs on the stable branch are merged or closed more quickly. Table 4 further highlights this point, showing that the mean and median times to complete code 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.

As shown in Table 4, 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, respectively. This is further illustrated in Figure 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 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

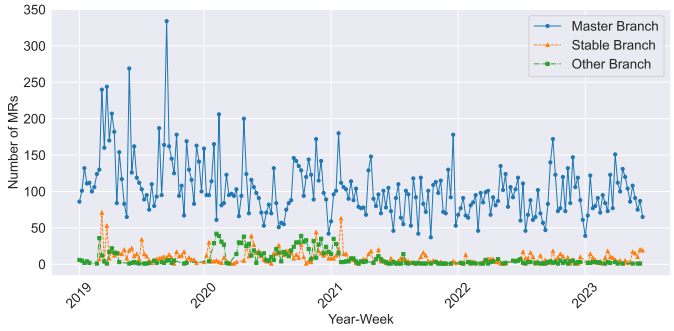

Fig. 9: Weekly MRs for Master, Stable, and Other branches

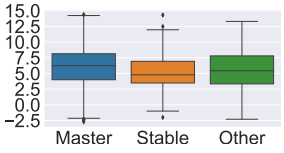

Fig. 10: Time to complete code review distribution across branches

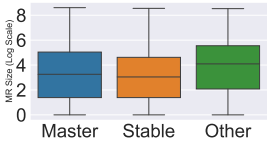

Fig. 11: MR Size Distribution across Branches

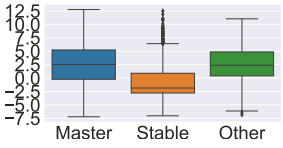

Fig. 12: Time to first comment distribution across branches

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 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.

Branch	Length	Time to complete code review Mean	Time to complete code 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

Table 4: Statistics for Master, Stable, and Other branches

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

Approach

Data Preparation

- **Data collection:** Similarly to Hasan et al. [8], we gathered data for each MR, including the discussions surrounding each MR. The collected metrics, indicated by a star ($\star$), are detailed in Table 2.
- **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. [8], who used the BoDeGha tool developed by Golzadeh et al. [7] 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 5 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 code 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 code review.
– **time to complete code review Analysis:** Explain the time to complete the review process 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 code review. Similarly to Zhang et al. [31], we train a mixed-effects linear regression model to explain time to complete code review. A mixed-effects regression model is appropriate as it accounts for fixed effects (MR characteristics impacting review time) and random effects (project-specific variations), enabling generalizable insights while minimizing project structure biases.
  We train a single global model using data from the four projects of our industrial partner, 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 code review, similarly to prior studies [8, 14]. This method evaluates the importance of independent variables without assuming interactions, filtering non-significant features and identifying key predictors using Chi-square (Chisq[8] = 0.05) statistics for robust analysis.
  • Study the impact of key metrics on time to complete code review using Accumulated Local Effects (ALE) plots, similarly to the approach taken by Khatoonabadi et al. [13]. A negative impact indicates shorter time to complete code review probability, while positive implies longer time to complete code review probability

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 6. Notably, the FPGA group shows the slowest response times, possibly due to

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

Table 5: Definition of used categories in this RQ

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

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 code 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 code 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 6: Time-to-first-comment (TFC) and review completion time (minutes).

Project	Time to Complete 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 7: 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 7, 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 9, 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 code reviews in a timely manner, and highlight their prevalence in accelerating the development workflow and team integration.

As shown in Table 8, the time to first human response is the most influential, explaining 93.97% of the variance in time to complete code review, indicating that faster human intervention to review the changes can statistically impact the overall time to complete code 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 code 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 code 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 code 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 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 code review process.

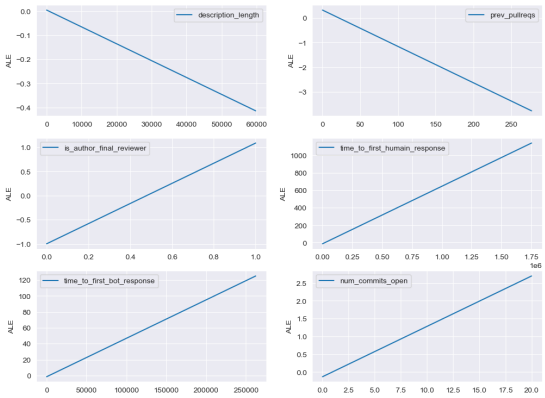

Fig. 13: Impact of the most important features on time to complete code review.

Table 8: Statistical analysis of features significantly impacting time to complete code 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 9: 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

5 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 code 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.

6 Conclusion and Practical Implications

This study underscores the pivotal role of MR data in offering comprehensive insights into multiple dimensions of the DevOps process, extending well beyond traditional code review analysis. By systematically examining MR data, we have gained a nuanced understanding of various aspects such as team collaboration, the impact of environmental and process changes, and the overall efficiency of the software development lifecycle.

Through the analysis of 26.7k MRs from a data center networking software company, our findings revealed that environmental changes, specifically the COVID-19 pandemic, led to a temporary increase in effort and a lasting shift in work patterns. These changes resulted in new habits that allowed greater flexibility with home working hours, with up to 70% of activities occurring outside regular office hours. Remarkably, despite these disruptions, productivity remained stable in terms of the number of MRs created and closed during the pandemic, highlighting the teams' adaptability and resilience.

The study also explored the impact of process changes, notably the migration to OpenShift technology. Initially, this transition caused fluctuations in times to complete code review and the prioritization of OpenShift-related tasks, which affected other development activities. However, as developers gained familiarity with the new technology, these metrics stabilized, reflecting successful integration and improved competency in using OpenShift.

Additionally, our research found that MRs on stable branches are consistently prioritized and resolved faster, emphasizing their significance in managing releases and addressing critical bug fixes. This prioritization illustrates the importance of efficient branch management strategies in maintaining a streamlined and reliable software delivery pipeline.

Our analysis of the code review process revealed that while automated tools (bots) play a crucial role in accelerating the initiation of code reviews by providing immediate feedback, it is the human reviewers who have the most substantial impact on reducing times to complete code review and enhancing the overall quality of code assessments. Human interactions were found to be essential in driving thorough code evaluations, with factors such as the experience of reviewers and the number of commits significantly influencing code review efficiency. This finding highlights the complementary roles of bots in expediting the review process and human reviewers in ensuring the depth and accuracy of the feedback provided.

Beyond its academic contributions, this study offers practical value for software teams and DevOps practitioners aiming to improve their processes. By leveraging MR data, organizations can gain insights into various dimensions of development beyond code re-

view—such as productivity, effort, and collaboration—as explored in this study. Additionally, MR data serves as a valuable resource for evaluating the impact of environmental and process changes on software activities, enabling comparisons across different periods or transitions (e.g., remote work shifts or infrastructure migrations). Because MR data is directly tied to developer's daily activity, it also offers a window into their engagement patterns and behaviour throughout the review process.

MR-based analysis further supports data-driven branch management. By examining review timelines, branch usage, and MR lifetimes—particularly on stable branches—organizations can prioritize releases more effectively, optimize integration workflows, and enhance bug resolution practices, ultimately leading to more reliable and maintainable software.

Moreover, leveraging ML can help organizations explain and predict key DevOps metrics. For instance, ML models can uncover the influence of mechanisms such as automated bot interactions, offering insight into how these tools affect efficiency. ML itself can also act as an enabler within the DevOps cycle, by supporting continuous analysis and intelligent feedback loops, making the process more adaptive and proactive.

For organizations adopting DevOps practices, these findings emphasize the strategic value of systematically analyzing MR data. Doing so enables teams to improve collaboration, automate repetitive tasks, and make informed decisions during the review process. Ultimately, MR-driven insights help ensure that DevOps workflows remain efficient, scalable, and responsive to the evolving demands of software development.

In the future, we aim to expand our analysis by investigating additional factors and events through MR data to assess its capability in capturing these changes. We also plan to extend our methodology across multiple organizations and contexts, exploring other mechanisms such as pull requests to evaluate how different review systems and contexts influence DevOps workflows.

References

[1] Eman Abdullah AlOmar. "Deciphering refactoring branch dynamics in modern code review: An empirical study on Qt". In: *Information and Software Technology* 177 (2025), p. 107596.

[2] Olga Baysal et al. "Investigating technical and non-technical factors influencing modern code review". In: *Empirical Software Engineering* 21 (2016), pp. 932–959.

[3] Amiangshu Bosu and Jeffrey C Carver. "Impact of developer reputation on code review outcomes in oss projects: An empirical investigation". In: *Proceedings of the 8th ACM/IEEE international symposium on empirical software engineering and measurement*. 2014, pp. 1–10.

[4] Alanna Brown et al. "state of DevOps report". In: *Puppet+ DORA* (2016).

[5] Moataz Chouchen et al. "Learning to Predict Code Review Completion Time In Modern Code Review". In: *Empirical Software Engineering* 28.4 (2023), p. 82.

[6] Yuanrui Fan et al. "Early prediction of merged code changes to prioritize reviewing tasks". In: *Empirical Software Engineering* 23 (2018), pp. 3346–3393.

[7] Mehdi Golzadeh et al. "A ground-truth dataset and classification model for detecting bots in GitHub issue and PR comments". In: *Journal of Systems and Software* 175 (2021), p. 110911.

[8] Kazi Amit Hasan et al. "Understanding the Time to First Response In GitHub Pull Requests". In: *arXiv preprint arXiv:2304.08426* (2023).

[9] Jez Humble and David Farley. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

[10] Khairul Islam et al. “Early prediction for merged vs abandoned code changes in modern code reviews”. In: *Information and Software Technology* 142 (2022), p. 106756.

[11] Ramtin Jabbari et al. “What is DevOps? A systematic mapping study on definitions and practices”. In: *Proceedings of the scientific workshop proceedings of XP2016*. 2016, pp. 1–11.

[12] Yujuan Jiang, Bram Adams, and Daniel M German. “Will my patch make it? and how fast? case study on the linux kernel”. In: *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE. 2013, pp. 101–110.

[13] SayedHassan Khatoonabadi et al. “On Wasted Contributions: Understanding the Dynamics of Contributor-Abandoned Pull Requests–A Mixed-Methods Study of 10 Large Open-Source Projects”. In: *ACM Transactions on Software Engineering and Methodology* 32.1 (2023), pp. 1–39.

[14] Alexandra Kuznetsova, Per B Brockhoff, and Rune Haubo Bojesen Christensen. “lmerTest package: tests in linear mixed effects models”. In: *Journal of statistical software* 82.13 (2017).

[15] Harsh Lal and Gaurav Pahwa. “Code review analysis of software system using machine learning techniques”. In: *2017 11th International Conference on Intelligent Systems and Control (ISCO)*. IEEE. 2017, pp. 8–13.

[16] Julien Legault. “An empirical investigation on the prevalence, impact, and manifestation of rework commits in the merge requests mechanism”. PhD thesis. École de technologie supérieure, 2022.

[17] Zhixing Li et al. “Redundancy, context, and preference: An empirical study of duplicate pull requests in OSS projects”. In: *IEEE Transactions on Software Engineering* 48.4 (2020), pp. 1309–1335.

[18] Chandra Maddila et al. “Nudge: Accelerating Overdue Pull Requests toward Completion”. In: *ACM Transactions on Software Engineering and Methodology* 32.2 (2023), pp. 1–30.

[19] Murtuza Mukadam, Christian Bird, and Peter C Rigby. “Gerrit software code review data from android”. In: *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE. 2013, pp. 45–48.

[20] Mala Murugappan and Gargi Keeni. “Blending CMM and Six Sigma to meet business goals”. In: *IEEE software* 20.2 (2003), pp. 42–48.

[21] Mahtab Nejati, Mahmoud Alfadel, and Shane McIntosh. “Code review of build system specifications: prevalence, purposes, patterns, and perceptions”. In: *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE. 2023, pp. 1213–1224.

[22] Cvetan Redzic and Jongmoon Baik. “Six sigma approach in software quality improvement”. In: *Fourth International Conference on Software Engineering Research, Management and Applications (SERA’06)*. IEEE. 2006, pp. 396–406.

[23] Leah Riungu-Kalliosaari et al. “DevOps adoption benefits and challenges in practice: A case study”. In: *Product-Focused Software Process Improvement: 17th International Conference, PROFES 2016, Trondheim, Norway, November 22-24, 2016, Proceedings 17*. Springer. 2016, pp. 590–597.

[24] Ian Sommerville and Jane Ransom. “An empirical study of industrial requirements engineering process assessment and improvement”. In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 14.1 (2005), pp. 85–117.

[25] Davide Spadini et al. “When testing meets code review: Why and how developers review tests”. In: *Proceedings of the 40th International Conference on Software Engineering*. 2018, pp. 677–687.

[26] Miroslaw Staron et al. "Measuring and visualizing code stability–a case study at three companies". In: *2013 Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement*. IEEE. 2013, pp. 191–200.

[27] Patanamon Thongtanunam et al. "Investigating code review practices in defective files: An empirical study of the qt system". In: *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE. 2015, pp. 168–179.

[28] Patanamon Thongtanunam et al. "Review participation in modern code review: An empirical study of the android, Qt, and OpenStack projects". In: *Empirical Software Engineering* 22 (2017), pp. 768–817.

[29] Rosalia Tufano et al. "Towards automating code review activities". In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE. 2021, pp. 163–174.

[30] Xunhui Zhang et al. "Pull request latency explained: An empirical overview". In: *Empirical Software Engineering* 27.6 (2022), p. 126.

[31] Yuanliang Zhang et al. "An evolutionary study of configuration design and implementation in cloud systems". In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE. 2021, pp. 188–200.

Authors

Samah Kansab is a Ph.D. candidate in Software Engineering at École de technologie supérieure (ÉTS), Montréal, Canada. Her research focuses on DevOps practices, software quality, and empirical software engineering. She collaborates with industrial partners to analyze software development processes using data-driven approaches, particularly in the context of networking software solutions.

Matthieu Hanania is a master's student at ÉTS. His research focuses on improving DevOps processes through data-driven analysis and automation. He collaborates on projects aimed at enhancing software development workflows, with a particular interest in ML for code review practices, and the impact of process changes on developer productivity.

Francis Bordeleau is a professor at ÉTS, specializing in software architecture, DevOps, and model-driven engineering. With extensive industry experience, he has worked on bridging the gap between academic research and real-world software engineering challenges, particularly in large-scale software development and deployment.

Ali Tizghadam is a senior technical expert at TELUS, Toronto, Canada. His work focuses on network intelligence, software-defined networking (SDN), and the intersection of AI with networking systems. With a strong background in telecommunications and network engineering, he leads research and development efforts to enhance the efficiency of modern network infrastructures.