Unveiling the Relationship Between Continuous Integration and Code Review: A Study with 10 Closed-source Projects

Ruben Silva
Federal University of Ceará
Quixadá, Brazil
rubensilva@ufc.br

Publio Silva
Federal University of Ceará
Quixadá, Brazil
publioufc@alu.ufc.br

Carla Bezerra
Federal University of Ceará
Quixadá, Brazil
carlailane@ufc.br

Anderson Uchôa
Federal University of Ceará
Itapajé, Brazil
andersonuchoa@ufc.br

Alessandro Garcia
Pontifical Catholic University of Rio de Janeiro
Rio de Janeiro, Brazil
afgarcia@inf.puc-rio.br

ABSTRACT

Companies have adopted Continuous Integration (CI) and Code Review (CR) as key practices to monitor and improve software quality continuously. These practices enable early detection and correction of issues and have shown promising results in open and closed-source projects. However, there is limited understanding of the relationship between CI and CR in closed-source environments. For instance, it remains unclear how CI practices influence CR (and vice versa) and to what extent bad CI practices hinder the CR process. To address this gap, we conducted a mixed-methods study involving ten closed-source projects developed by industry partners. We collected and analyzed 32 metrics related to CI and CR. In addition to the quantitative analysis, we gathered perceptions from project teams to (i) validate the identified correlations, (ii) explore the effects of bad CI practices on CR, and (iii) understand the perceived benefits and challenges of using CI and CR together. Our results revealed 23 correlations between CI and CR processes, leading to the following key findings: (i) the workload and execution time of CI can influence code review time; (ii) CI and CR execution times are more correlated when the processes occur sequentially; (iii) bad CI practices (such as long-lived branches, poor testing strategies, or complex build schemes) can negatively impact CR, causing delays, reviewer overload, and undetected issues; (iv) CI adds value to CR by anticipating problems, reducing manual tasks, and supporting the distribution of updates; and (v) the joint use of CI and CR presents challenges, such as ensuring code quality, resolving merge conflicts, and aligning processes. These findings shed light on the interplay between CI and CR and offer insights to improve their combined use in closed-source software development.

KEYWORDS

continuous integration, code review, closed-source projects

1 Introduction

In modern software development, code review (CR) and continuous integration (CI) are widely adopted in both open-source and industrial projects to enhance the development process [30, 46, 48, 51, 52]. These practices help ensure software quality and timely delivery. CR focuses on detecting and correcting issues introduced during development [2, 29, 44, 47]. It typically involves two or more developers reviewing code changes submitted by a peer and reporting issues for correction [2, 36]. Tools such as GitLab and Gerrit support CR processes. CI enables developers to frequently integrate code into the main codebase through automated procedures [15, 21, 26], supported by tools like Travis and GitLab CI.

Together, CR and CI form a central pillar of DevOps practices, enabling teams to balance fast-paced delivery with sustained code quality. Their integration is increasingly seen as a vital step toward modern, scalable engineering workflows [15].

Despite their benefits, both CI and CR practices continue to face persistent challenges. For CI, common problems include: poor dependency management leading to overly complex build schemes [9, 25]; increased build times [22]; inadequate testing strategies [7, 40]; and long-lived commits or branches causing integration conflicts [9, 12, 25].For CR, challenges include: delays in review completion due to prioritization issues or overloaded reviewers [23, 37]; lack of documentation or overly large changesets [27]; and incomplete or unrecorded feedback [14].

While the challenges of CI and CR are well documented, few studies have explored their relationship [5, 35, 48, 51]. Existing research has primarily focused on non-technical factors such as reviewer workload, developer participation, and patch size [5], or on how CI influences merge request discussions [51]. Most of these studies are based on open-source projects [5, 48, 51], with limited investigation into industrial settings [12, 51].

To address these gaps, we investigate the relationship between CI and CR processes, seeking to identify correlations and understand how code reviewers perceive their interplay. Given the existence of documented CI bad practices in software development [17, 20, 33], we also explore their potential negative effects on CR, along with the perceived benefits and challenges of integrating CI and CR.

We conducted a mixed-methods study (quantitative and qualitative) across ten closed-source projects from our industrial partners. We collected 32 different metrics and complemented the quantitative analysis with focus group discussions, addressing: (1) validation of identified correlations; (2) the impact of CI bad practices on CR; and (3) the benefits and challenges of CI-CR integration.

Our results reveal 23 correlations between CI and CR processes, leading to the following key findings: (1) CI workload and execution time influence CR review time; (2) CI and CR execution times are more correlated when processes occur sequentially; (3) CI bad practices delay CR, increase workload, and raise the risk of undetected issues; (4) CI adds value to CR by anticipating problems, streamlining update distribution, and reducing manual tasks; and (5) although CI and CR integration offers benefits, it also introduces challenges that require adaptive and strategic management to optimize software development.

2 Background and Related Work

Continuous Integration. CI is a process of modern software engineering that is based on the frequent integration of code by developers [11]. A common CI process is usually composed of mechanisms that automatically allow the execution of tests, static analysis, and the system's construction. These mechanisms make it possible for errors to be identified early, causing the reduction of system errors in the production instance [16].

In practice, CI performs as follows: (i) the moment the developer submits a set of changes to the version management server, the CI server takes action to proceed with the build check; (ii) depending on the configuration chosen, the execution of the CI can perform tests (e.g., front-end, back-end, unitary, integration, among others); (iii) a static analysis is performed to verify the syntactic quality of the code; and, (iv) finally with the approved code, the changes are integrated into the main development line [30]. Figure 1 shows how a CI pipeline works.

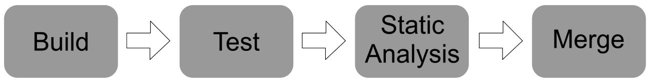


Figure 1: Overview of the Continuous Integration Pipeline

CI is commonly used in conjunction with the CR to enhance the review process. This is because a well-structured CI pipeline can provide reviewers with more in-depth insights into the set of changes that have been submitted to the review [35]. In addition, a good CI pipeline provides developers with the possibility to refactor the source code even before the modifications reach the reviewers. That way, the review won't have to stick to the more complex details of the changes, leaving aside basic implementation aspects [31].

Modern Code Review. In general, CR is limited to inspecting the code for defects. Fagan [13] formalizes a well-structured process for CR based on line-by-line manual review. However, many companies today prefer a more lightweight process for CR, called Modern Code Review (MCR). In MCR, the reviews are performed more informally, and in most cases, supported by tools, such as Gerrit and GitLab [2]. Figure 2 shows the steps commonly used in the CR process [32].

In summary, the code review process follows this steps: (1) a developer submits a set of changes to be revised by the reviewers; (2) then, according to their experience, the reviewers provide a set of comments and improvement suggestions; (3) after the correction of the issues raised by the reviewers, the developer submits a new set of changes to be revised; and (4) the reviewers need to reach an agreement about the approval of the code changes to be integrated into the codebase. This cycle is repeated until the reviewers reach an agreement to approve or reject the proposed code change.

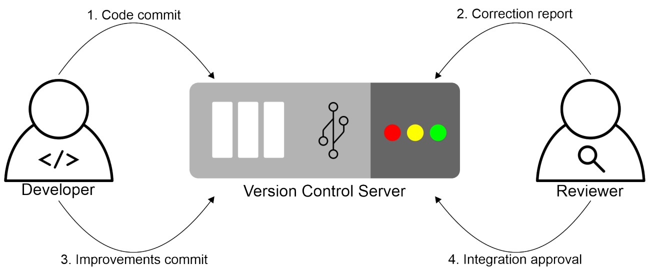


Figure 2: Overview of the Code Review Process

Continuous Integration Bad Practices. Duvall [10] alert to the fact that the CI needs to be well implemented to obtain the benefits and structure a series of bad practices related to the misuse of CI. According to Zampetti et al. [53], some CI bad practices concern (i) deployment of artifacts that have been generated in a local environment, and (ii) deployment without a previous verification in a representative environment.

Duvall [9] initially created a catalog with 50 patterns (and their corresponding antipatterns) regarding several phases or relevant topics of the CI process. Zampetti et al. [53] derived a catalog of 79 CI bad practices organized into 7 categories spanning across the different dimensions of CI pipeline management: Repository, Build Process Organization, Quality Assurance, Delivery Process, Infrastructure Choices, Build Maintainability, and Culture. In this work, we use Zampetti's CI bad practices catalog [53].

2.1 Related Work

Although several studies have investigated CI and CR individually, relatively few have focused on understanding the relationship between these two practices [5, 35, 48, 51].

Studies exploring the interplay between CI and CR. Rahman and Roy [35] analyzed thousands of automated build logs from GitHub open-source projects to explore if there was a correlation between CI status and aspects of CR, such as review participation and review quality. Their findings indicated that successful CI builds tend to encourage greater reviewer engagement in pull requests. Zampetti et al. [51] conducted a qualitative analysis of integration requests across 69 open-source projects and observed that pull requests with successful CI builds were more likely to be merged into the main codebase. Cassee et al. [5] performed an exploratory study and found that after the adoption of CI, pull requests experienced fewer discussion comments, suggesting that automated checks might have reduced the need for extended human communication. Wessel et al. [48] examined the adoption of CI bots that post comments in pull requests and identified an increase in merged requests, a decrease in non-integrated requests, and a reduction in communication volume among development team members.

Studies examining how CI affects CR processes. Maipradit et al. [28] analyzed the use of the `recheck` command in the OpenStack community and found that repeated CI builds, although common, rarely changed the build status but significantly delayed code reviews and consumed substantial computational resources. Silva et al. [42] emphasized the role of effective code review practices in detecting defects and improving code maintainability, indirectly suggesting that enhancements in CI processes could positively influence CR outcomes. Bernardo et al. [3] evaluated the impact of TravisCI adoption across open-source projects, revealing that while CI improved developer confidence and decision-making, it did not necessarily accelerate the delivery of pull requests. Santos et al. [38] introduced Holter, a tool designed to offer deeper insights into CI maturity and practices by continuously tracking key CI-related metrics, aiming to improve adoption and effectiveness. Kudrjavets and Rastogi [24] explored practitioners' perceptions regarding review speed, highlighting the importance of fast CI feedback and rapid review cycles to enhance team productivity and satisfaction.

Studies that used CI and CR metrics. Chen et al. [6] analyzed 190 pull requests from 142 GitHub projects. They developed a tool to predict whether a code change would be integrated, based on a combination of quantitative metrics and developers' qualitative opinions about the change. In our study, we selected a subset of 20[1] of the metrics used by Chen et al. [6]. We implemented them in a script for automatic data collection in closed-source projects hosted on GitLab. We focused on analyzing aspects related to team participation in CRs and characteristics of the CI process within integration requests. Paixao et al. [34] tackled the issue that most code repositories do not allow review comments to be properly linked to the respective system versions at the time of review. To address this, they developed CROP ("**C**ode **R**eview **O**pen **P**latform"), a platform that links CRs to complete system snapshots corresponding to the review moment. Similarly, in this work, we used a software tool to extract CR metrics systematically from code review data.

While these works provide important insights, our study introduces several significant advancements. First, we focus exclusively on closed-source industrial projects, addressing a context that remains largely underrepresented in the existing literature, which predominantly concentrates on open-source ecosystems. Second, we combine a rigorous quantitative analysis of 32 different CI and CR metrics with a qualitative investigation based on real perceptions of development teams involved in the studied projects. Unlike previous studies, we explicitly examine the impact of CI bad practices on CR, exploring how inefficient CI configurations and workflows can adversely affect review quality and turnaround times.

Furthermore, rather than limiting our investigation to isolated technical factors, we adopt a holistic perspective that considers technical, organizational, and human aspects shaping the relationship between CI and CR processes. Our findings not only reinforce existing knowledge but also extend it by providing practical insights and lessons learned from real-world industrial scenarios. Thus, our work substantially advances the understanding of how CI and CR interact in practice, offering both researchers and practitioners a deeper view of these crucial software engineering practices.

3 Study Settings

This study aims to investigate the relationship between Continuous Integration (CI) and Code Review (CR) processes. We combine CI and CR metrics, with team perceptions involved in industrial projects to identify: (1) how team members perceive correlations between CI and CR processes, (2) the impact of CI bad practices on the CR process, and (3) the perceived benefits and challenges of using CI and CR together. As a quantitative/qualitative research effort, this study is restricted to the context of closed-source projects. We detail each research question (RQ) as follows.

RQ$_1$: *Which aspects of the CI and CR processes are correlated?* This research question aims to identify correlations between various aspects of CI and CR processes in industrial software projects. By exploring these correlations, we aim to understand whether and how these two software engineering practices are connected in practice. The results from RQ$_1$ serve as the foundation for RQ$_2$.

RQ$_2$: *What are the development teams' perceptions regarding the correlation between CI and CR processes?* This question seeks to investigate how the teams from the analyzed projects perceive the correlations identified in RQ$_1$. Understanding their perspectives helps validate our findings and uncover implicit details about CI and CR practices within the partner organizations. It also allows us to identify potential flaws or inefficiencies that may have influenced the observed correlations.

RQ$_3$: *How do CI bad practices affect the CR process?* This question aims to explore how development teams perceive the effects of CI bad practices on CR. The bad practices considered in this study were identified in a previous analysis within the same industrial context [43]. Our goal is to determine whether such practices negatively impact the CR process beyond CI's own efficiency.

RQ$_4$: *What are the benefits and challenges of using CI in the CR process?* RQ$_4$ aims to understand how the outcomes produced by the CI process are leveraged during CR. Building on the previous RQs, RQ$_4$ provides a focused analysis of the code reviewers' experiences in projects that integrate both CI and CR, shedding light on how CI may enhance or hinder the CR process.

3.1 Study Steps and Procedures

We describe the steps to support our investigation as follows.

Step 1: Selecting closed-source systems for analysis. Our study focuses on closed-source industrial projects. This choice was motivated by the desire to capture CI and CR dynamics in controlled, real-world environments where practices are often more formalized and traceable. Unlike many open-source repositories, these projects may offer structured CI pipelines, regulated review processes, and access to team members, allowing for richer qualitative insights.

From the 46 projects developed by our industrial partners, we selected 10 projects based on the following criteria: (i) systems that have been configured for CI for at least one year; and (ii) systems that have been in operation for at least one year. These criteria were applied to ensure a sufficiently broad timeframe for analysis, which is necessary to obtain consistent and meaningful results. Table 1 presents general data for each selected system. We omitted the system names due to intellectual property restrictions. The first column lists the system, followed by: system domain; number of versions (NV); lines of code (LOC); number of pipelines (NP); and number of merge requests (NMR).

It is important to highlight that all selected systems from our industrial partners adopt the CR process as part of their development workflow. Another key characteristic is that our partners started using CI approximately three years ago, suggesting that they have acquired a moderate level of experience with this practice.

[1]In total, we implemented 28 metrics: 20 extracted from the literature and 8 defined specifically for our context.

Table 1: General data about the analyzed systems

System	Domain	NV	LOC	NP	NMR
S1	Skills-based Organizational Management	1	39,585	1,764	367
S2	Student Assistance	1	22,391	731	614
S3	Foreign Language Reading Proficiency	20	18,098	2,135	224
S4	Management of Complementary Activities for Graduating Degrees	1	3,209	505	85
S5	Storeroom Management	4	22,045	106	290
S6	Organizational Risk Management	1	20,336	818	154
S7	Social Project Management	1	17,583	303	107
S8	Electronic Dental Records	7	65,636	1,218	443
S9	Academic Events Management	12	26,724	255	381
S10	Open Data Mining	2	4,092	364	147

Step 2: Selecting CI and CR metrics for analysis. We used the work of Chen et al. [6] as the basis for selecting the metrics in this study. This study summarizes a widely used set of metrics extracted from previous research [5, 6, 18, 19, 45, 50, 54]. Additionally, we selected metrics that met the following criteria: (1) the metrics should have a direct relationship with CI or CR, and (2) the metrics should be collectible within the GitLab environment. Based on these criteria, we selected a total of 24 metrics: 19 related to CR and 5 related to CI. Furthermore, to capture more detailed aspects of the CI and CR processes, we defined eight additional metrics related to the following: errors reported by CI, review duration, and the amount of work performed by CI, resulting in a total of 32 metrics.

Table 2 presents an overview of the 32 selected metrics, grouped by category (CI and CR) and by dimension (time, participation, and intensity). These dimensions help measure specific details of the CI and CR processes in software projects: the *time* dimension allows us to understand the duration of events associated with each process (CI or CR); the *participation* dimension provides insight into how team members engage in these processes; and finally, the *intensity* dimension gives an idea of the workload involved in CI and CR. It is worth noting that, among the metrics belonging to the CI category, none relate to the participation dimension, as CI processes are automated and therefore do not involve direct interaction from project team members.

Step 3: Collecting and validating the computation of the target CI and CR metrics for analysis. To automate the collection of the target metrics, we developed a Python script using the Python-GitLab library[2] to facilitate obtaining data, since our target systems are hosted on the GitLab platform. The workflow of the script can be described as follows: (1) first, authentication with the GitLab API takes place; (2) next, the primary data of the selected projects is retrieved; (3) for each project, the set of all *merge requests* of the analyzed projects is recovered; (4) for each *merge request*, all metrics are computed; and, finally, (5) the data is stored in CSV format, generating one file per project.

The data collection through our script returned a dataset consisting of 942 merge requests extracted from ten closed-source projects hosted on GitLab. The collection was carried out individually for each project, resulting in several CSV files. To ensure the reliability of the collected data, we performed a validation of the metric values. One of the study members manually inspected the data to identify anomalies in the metrics computed by the script. As a result, it was necessary to implement adjustments to the script code.

In our context, we use the term build to refer to each execution of a CI pipeline triggered by a merge request or commit. A pipeline corresponds to a full integration workflow composed of one or more jobs, which are the individual execution steps (e.g., test, lint, deploy). Thus, the reported build counts reflect all development-related pipeline executions, including those triggered by updates, re-runs, or minor changes, not only staging or production releases. For instance, in system S4, the high number of builds (505) is due to frequent updates and re-executions of pipelines during its development cycle, despite its small codebase and single version.

While some of the extracted metrics leverage GitLab-specific attributes, the conceptual definitions are generalizable and can be derived from comparable features in other CI/CR platforms (e.g., GitHub, Bitbucket). Thus, our approach can be adapted to different environments as long as the necessary metadata is accessible.

Step 4: Conducting the data analysis. We applied statistical methods to identify correlations between the factors considered in our study. In this context, the data obtained from Step 3 were analyzed from two perspectives: (i) overall, aggregating all projects; and (ii) separately for each project. To standardize the metric values, we applied data normalization using "MinMaxScaler" method from the Python "sklearn" library [4]. This method requires lower and upper bounds to be set as parameters for normalization; we chose -1 and 1. Next, we conducted an outlier analysis to avoid potential bias in our dataset. Figure 3 presents the overall distribution of (a) CI metrics and (b) CR metrics through boxplots.

We observed a significant number of outliers in both CR and CI cases, which could potentially influence the correlation analysis. Consequently, we removed the outliers using the Tukey test [8]. However, after generating the correlation matrix, we did not observe significant changes in the correlation coefficient values. Thus, we opted to include the outliers in the metric analysis, as our dataset did not contain a substantial amount of data.

Finally, to perform the correlation analysis, we first analyzed the distribution of the data across all metrics (both CI and CR). The distribution plots are available in our replication package. We observed that the data did not follow a normal distribution. Consequently, we computed Spearman's rank correlation coefficient [39], using a 95% confidence interval (ρ-value ≥ 0.05). The Spearman's correlation coefficient ranges from -1 to +1, where 0 indicates no correlation, and values closer to -1 or +1 indicate stronger correlations between the variables [39]. To facilitate the interpretation of the correlation coefficients, we used the following scale: (1) *very strong correlation* (0.8 to 1.0 or -0.8 to -1.0); (2) *strong correlation* (0.6 to 0.8 or -0.6 to -0.8); (3) *moderate correlation* (0.4 to 0.6 or -0.4 to -0.6); (4) *weak correlation* (0.2 to 0.4 or -0.2 to -0.4); and (5) *very weak or no correlation* (0.0 to 0.2 or 0.0 to -0.2). Finally, we analyzed the significant correlations to provide consistent interpretations.

Step 5: Collecting of code reviewers' perceptions on correlations, CI bad practices, and benefits/challenges of CI and CR. To collect the perceptions of specialists who acted as code reviewers in the projects analyzed in our study, we employed a focus group approach to address our last three research questions. The planning and execution of the focus group were based on the methodology proposed by Almeida et al. [1], and involved the following steps:

Step 1: Environment preparation. First, we selected tools to support the focus group. We used a virtual whiteboard tool, Mural[3], to

[2]https://python-gitlab.readthedocs.io/en/stable/

[3]https://www.mural.co/

Table 2: Metrics characterization

Dimension	ID	Metric	Description	Source
			Code Review	
Time	M1	Review Time	Measures the time from merge request creation to merge, reflecting the full review cycle regardless of CI status.	[19]
	M2	Mean Time Between Comments	Shows interaction patterns during review. A short interval indicates quick back-and-forth, while a long interval may signal delays or low engagement.	[50]
	M3	Time Between First and Last Comment	Captures review duration from the first to the last comment. Provides an overview of how long reviewers discuss a given code change.	This Work
Participation	M4	Number of Participants	Counts distinct reviewers in the discussion. More participants can improve diversity of feedback and code quality.	[18, 19]
	M5	Inline Comments	Number of comments inserted directly into the source. High counts may signal problematic or complex code areas.	[5, 6]
	M6	General Comments	Comments made in discussion threads rather than inline. Often cover higher-level topics like design or architecture.	[5, 6]
	M7	Effective Comments	Comments that lead to code changes or highlight real issues. Indicative of valuable review feedback.	[5, 6]
	M8	Total Comments	Overall number of comments in a review. Reflects discussion intensity and reviewer engagement.	[18, 19, 45, 50]
	M9	First Response Time	Time from review creation to the first reviewer comment. Assesses reviewer promptness in providing initial feedback.	[6, 50]
	M10	Mention Count	Number of "@" mentions in comments. Helps track individual reviewer involvement or attention to specific code parts.	[54]
Intensity	M11	Self-merged Commits	Commits where the author and the merger are the same person. Such commits may bypass peer validation, potentially increasing the chance of undetected issues.	[50]
	M12	Source Lines Added	Lines of production code added. Indicates development effort and potential review scope.	[6, 18, 19, 50]
	M13	Source Lines Removed	Lines of production code deleted. Can reflect refactoring or bug fixes, but also risk if not reviewed carefully.	[6, 18, 19, 50]
	M14	Test Lines Added	Lines of test code added. Reflects testing effort and may affect review complexity.	[18, 19]
	M15	Test Lines Removed	Lines of test code deleted. Indicates test refactoring or coverage changes.	[18, 19]
	M16	Number of Reviews	Count of distinct review sessions over a period. Useful for trend analysis of review activity.	[6, 18, 19, 45, 50]
	M17	Files Added	Number of new files created. Higher counts can increase review scope.	[19]
	M18	Files Deleted	Files removed during development. Impacts codebase integrity and review needs.	[19]
	M19	Files Modified	Number of files changed. Ensures modifications are properly reviewed to avoid regressions.	[19]
	M20	Total Changes	Sum of added, deleted, and modified files. Gauges overall work size and review effort.	[18, 19, 45]
			Continuous Integration	
Intensity	M21	Total Builds	Number of CI pipeline runs. Reflects build frequency and computational effort.	This Work
	M22	Successful Builds	Count of CI runs that passed. Indicates code stability and readiness for deployment.	This Work
	M23	Failed Builds	Count of CI runs that failed. Highlights build issues needing investigation.	This Work
	M24	Total Jobs	Number of individual CI jobs executed. Measures computational workload per pipeline.	This Work
	M25	Successful Jobs	Jobs that completed successfully. Reflects reliability of CI tasks.	This Work
	M26	Failed Jobs	Jobs that failed during CI. Points to flaky tests or configuration errors.	This Work
	M27	Tests Run	Total tests executed in CI. Important for coverage and quality assessment.	[41]
	M28	Tests Failed	Number of tests that failed. Indicates functional issues requiring fixes.	[41]
	M29	Tests with Errors	Tests that errored (e.g., exceptions). Points to logic or configuration problems.	[41]
	M30	Skipped Tests	Tests not executed. May indicate environment or dependency issues.	[41]
Time	M31	CI Latency	Time from code change to CI completion. High latency can slow development and review cycles.	[6, 50]
	M32	Build Correction Interval	Time between a failed build and the next successful one. Measures efficiency in diagnosing and fixing CI failures.	This Work

facilitate the visualization of the discussion topics and to encourage the collaboration of participants. Within the platform, interactive boards were created based on our research questions, where participants (1) voted and expressed their agreement or disagreement with two statements derived from the results of the first research question, (2) listed potential effects of CI bad practices on the code review process, and (3) identified benefits and challenges of CI in CR. Figure 4 summarizes the boards created. The sessions were held via Google Meet, and recorded and transcribed using the tldv tool[4] to facilitate analysis.

Step 2: Participant selection. We invited specialists who were part of the development teams of the studied systems and who had acted as code reviewers in the analyzed projects to participate in the focus group. Before the session, a questionnaire was applied to characterize the participants in terms of (1) education level, (2) experience with CI, and (3) experience with CR. Table 3 presents the participants' profiles.

Table 3: Reviewers characterization

ID	Education level	CI expertise	CR expertise
P1	Specialization	Basic	Intermediate
P2	Master's Degree	Basic	Basic
P3	Doctorate Degree	Advanced	Advanced
P4	Doctorate Degree	Intermediate	Advanced

Step 3: Focus group execution. The focus group was conducted remotely via Google Meet and lasted approximately one hour. The session followed this structure: (1) introduction of the research topic, explanation of the session dynamics, and clarification of participants' questions; (2) discussion on the influence of CI on CR based on two statements evaluated on a scale from "strongly disagree" to "strongly agree", with participants elaborating through oral discussion and shared notes; (3) analysis of the effects of 14 previously identified CI bad practices [43] on CR, where participants discussed and documented possible impacts; (4) after a 5-minute informal break, a brainstorming session was held to gather perceived benefits and challenges of CI in CR, again using shared notes; and (5) conclusion with a thank-you message and summary of the discussion. The session was structured to keep participants focused and avoid unrelated topics.

Step 4: Analysis of results. Finally, we analyzed the results based on participants' perceptions. We employed qualitative analysis of the shared notes and complemented it with participants' statements made during the session.

[4]https://tldv.io/

4 Results and Discussions

4.1 Related Aspects Between CI and CR Processes (RQ_1)

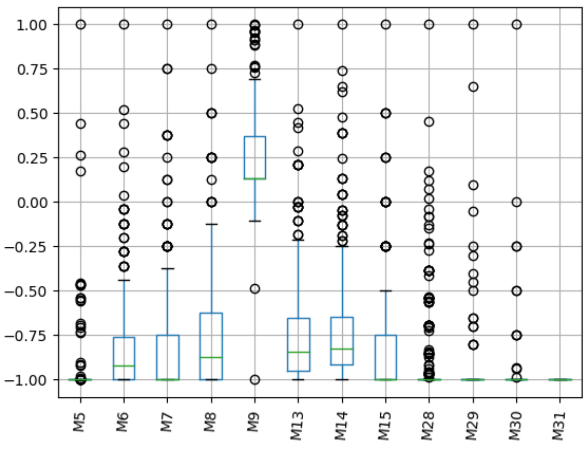
To answer $\mathbf{RQ_1}$, we analyzed the strongest correlations (i.e., coefficients greater than 0.6) between the CI and CR metrics. These relationships are visualized in Figure 5, which presents a heatmap-style correlation matrix with color gradients indicating the strength of each correlation, and are further detailed in Table 4.

Table 4 overviews the strongest correlations observed across individual projects and the overall dataset, distinguishing between direct (positive) and inverse (negative) relationships. Key correlations and their possible implications are summarized below:

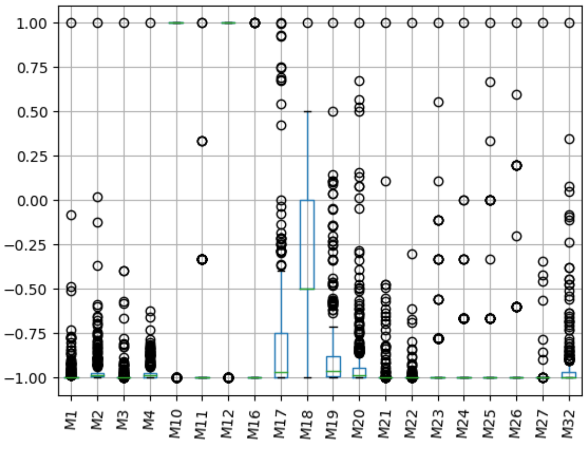
- **CI Latency** shows strong positive correlation with **Review Time**, indicating potential workflow delays, while it inversely correlates with **Number of Participants** and **Files Added** – possibly reflecting simpler, smaller changes in faster cycles.
- **Failed Builds** are positively correlated with **Review Time** and **Files Modified** (in some projects), and inversely correlated with **Self-Merged Commits**, suggesting that code quality or review bypasses may impact outcomes. **Number of Participants** appears to mitigate failures.

Table 4: Summary of the strongest CI-CR correlations observed across individual projects and the overall dataset

CI	CR	General	22	26	36	39	41	54	55	56	78	133	Occurrences
CI Latency	Review Time	0.79	0.93	0.82	0.83			0.87	0.88	0.78		0.63	**8**
	Number of Participants				0.71			0.85	0.71				**3**
	Files Added							-0.62					**1**
Failed Builds	Review Time							0.79	0.62	0.72			**3**
	Files Modified			0.74			-0.68						**2**
	Number of Participants							0.65					**1**
	Self-merged Commits			-0.68									**1**
Failed Jobs	Number of Participants						0.82						**1**
	Review Time						0.8						**1**
Successful Builds	Files Modified			-0.68									**1**
	Self-merged Commits			0.67									**1**
Successful Jobs	Review Time							0.81	0.7	0.72			**3**
	Files Modified						-0.77						**1**
	Number of Participants							0.84					**1**
	Source Lines Added						-0.65						**1**
Tests Run	Review Time		0.63										**1**
Total Builds	Review Time	0.67	0.61	0.67				0.86	0.75	0.7			**6**
	Number of Participants							0.84	0.64				**2**
	Files Added							-0.6					**1**
	Files Modified						-0.68						**1**
Total Jobs	Review Time	0.68					0.7	0.86	0.76	0.69			**5**
	Number of Participants							0.8	0.61				**2**
	Files Added							-0.63					**1**
Occurrences		**3**	**3**	**6**	**2**	**0**	**7**	**13**	**8**	**5**	**0**	**1**	**48**



(a) General boxplot diagram for the CI metrics



(b) General boxplot diagram for the CR metrics

Figure 3: CI/CR metrics dist. (norm., outlier detection)

- **Failed Jobs** correlate with longer **Review Time** and benefit from higher **Participant** involvement, following patterns similar to failed builds.
- **Successful Builds** inversely correlate with **Files Modified** but positively with **Self-Merged Commits**, potentially reflecting contributor experience or trusted changes.
- **Successful Jobs** are associated with shorter **Review Times** and increased **Participant Collaboration**, but inversely correlated with **Files Modified** and **Source Lines Added**, indicating sensitivity to larger or riskier changes.
- **Tests Executed** correlate with longer **Review Time**, likely due to validation effort.
- **Total Builds** positively correlate with **Review Time** and **Participant Count**, but inversely with **Files Added** and **Files Modified** – perhaps due to batch testing or integration thresholds.
- **Total Jobs** follow similar patterns, correlating with **Review Time** and **Participants**, but showing inverse trends with **Files Added**.

By analyzing Table 4, we observed that all CI and CR metric dimensions listed in Table 2 show at least one strong correlation. Notably, CI execution time (measured by CI Latency) and CI workload (Total Builds, Total Jobs, Failed Builds, Successful Jobs) strongly correlate with code review execution time (Review Time). This is particularly relevant given that CI and CR typically occur in parallel, and reviewers are expected to begin their manual analysis only after CI completion. Furthermore, CI Latency and Review Time emerged as the most frequent metrics among the correlations, reinforcing the idea that CI duration may influence CR duration.

In addition, we noticed that successful builds and jobs tend to correlate positively with self-merged commits and reviewer participation, whereas failed builds are more frequently associated with broader file modifications and longer review times. This contrast may reflect differences in integration confidence: smaller or trusted contributions flow more smoothly, while larger or complex changes require more effort and are prone to issues.

Figure 4: Focus group mural board

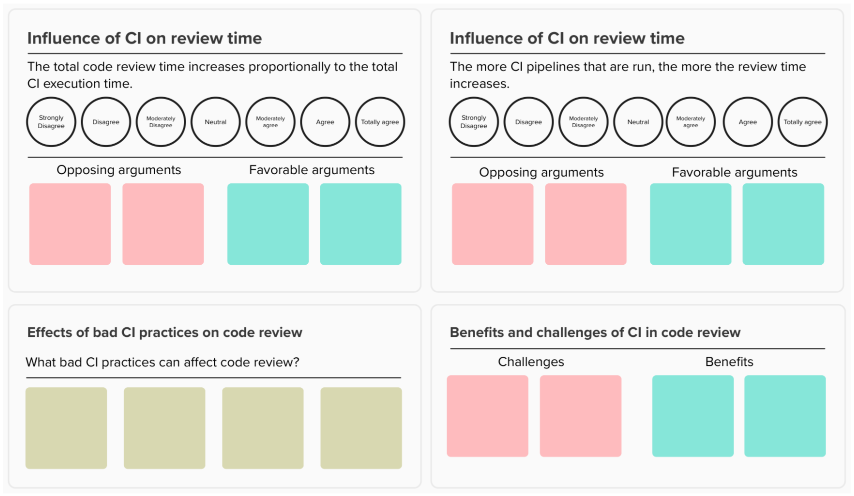
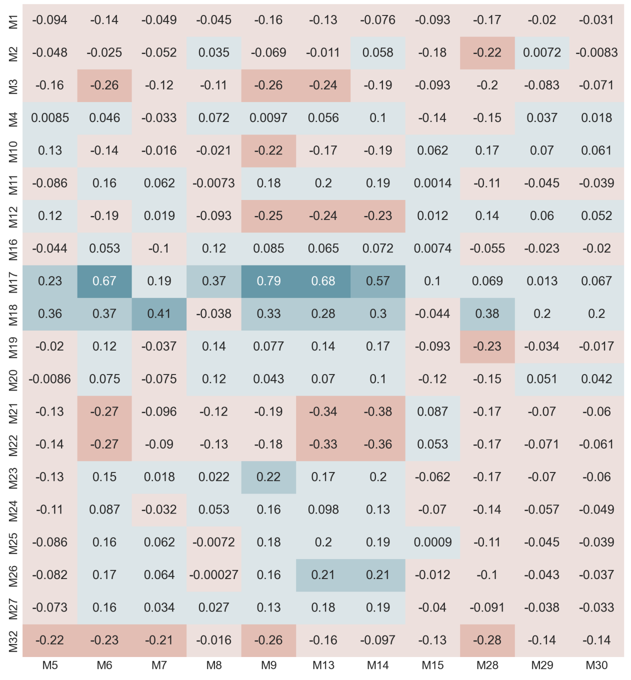

Influence of CI on review time
The total code review time increases proportionally to the total CI execution time.
Strongly Disagree
Disagree
Moderately Disagree
Neutral
Moderately agree
Agree
Totally agree
Opposing arguments
Favorable arguments
Influence of CI on review time
The more CI pipelines that are run, the more the review time increases.
Strongly Disagree
Disagree
Moderately Disagree
Neutral
Moderately agree
Agree
Totally agree
Opposing arguments
Favorable arguments
Effects of bad CI practices on code review
What bad CI practices can affect code review?
Benefits and challenges of CI in code review
Challenges
Benefits

Figure 5: Correlation matrix between CI and CR metrics



	M5	M6	M7	M8	M9	M13	M14	M15	M28	M29	M30
M1	-0.094	-0.14	-0.049	-0.045	-0.16	-0.13	-0.076	-0.093	-0.17	-0.02	-0.031
M2	-0.048	-0.025	-0.052	0.035	-0.069	-0.011	0.058	-0.18	-0.22	0.0072	-0.0083
M3	-0.16	-0.26	-0.12	-0.11	-0.26	-0.24	-0.19	-0.093	-0.2	-0.083	-0.071
M4	0.0085	0.046	-0.033	0.072	0.0097	0.056	0.1	-0.14	-0.15	0.037	0.018
M10	0.13	-0.14	-0.016	-0.021	-0.22	-0.17	-0.19	0.062	0.17	0.07	0.061
M11	-0.086	0.16	0.062	-0.0073	0.18	0.2	0.19	0.0014	-0.11	-0.045	-0.039
M12	0.12	-0.19	0.019	-0.093	-0.25	-0.24	-0.23	0.012	0.14	0.06	0.052
M16	-0.044	0.053	-0.1	0.12	0.085	0.065	0.072	0.0074	-0.055	-0.023	-0.02
M17	0.23	0.67	0.19	0.37	0.79	0.68	0.57	0.1	0.069	0.013	0.067
M18	0.36	0.37	0.41	-0.038	0.33	0.28	0.3	-0.044	0.38	0.2	0.2
M19	-0.02	0.12	-0.037	0.14	0.077	0.14	0.17	-0.093	-0.23	-0.034	-0.017
M20	-0.0086	0.075	-0.075	0.12	0.043	0.07	0.1	-0.12	-0.15	0.051	0.042
M21	-0.13	-0.27	-0.096	-0.12	-0.19	-0.34	-0.38	0.087	-0.17	-0.07	-0.06
M22	-0.14	-0.27	-0.09	-0.13	-0.18	-0.33	-0.36	0.053	-0.17	-0.071	-0.061
M23	-0.13	0.15	0.018	0.022	0.22	0.17	0.2	-0.062	-0.17	-0.07	-0.06
M24	-0.11	0.087	-0.032	0.053	0.16	0.098	0.13	-0.07	-0.14	-0.057	-0.049
M25	-0.086	0.16	0.062	-0.0072	0.18	0.2	0.19	0.0009	-0.11	-0.045	-0.039
M26	-0.082	0.17	0.064	-0.00027	0.16	0.21	0.21	-0.012	-0.1	-0.043	-0.037
M27	-0.073	0.16	0.034	0.027	0.13	0.18	0.19	-0.04	-0.091	-0.038	-0.033
M32	-0.22	-0.23	-0.21	-0.016	-0.26	-0.16	-0.097	-0.13	-0.28	-0.14	-0.14

Finding 1: CI workload and execution time influence code review duration, with smoother CI cycles linked to trusted contributions and heavier workloads to complex changes.

4.2 Development Teams' Perceptions of CI-CR Correlation (RQ2)

To address **RQ$_2$**, we analyzed the teams' perceptions on correlations identified in RQ$_1$. We summarized these correlations into two statements: (i) *"the total code review time increases proportionally to the total CI execution time"*, and (ii) *"the more CI pipelines are executed, the more the review time increases."* Participants discussed each statement during the focus group, expressing their agreement levels and providing supporting or opposing arguments, as follows.

The total code review time increases proportionally to the total CI execution time. Voting results showed a tendency toward disagreement: two participants chose *"moderately disagree"*, while one selected *"neutral"* and another *"moderately agree"*. At first glance, this could suggest a contradiction with the strong correlations identified in RQ$_1$. However, participants' comments revealed deeper insights. For instance, #P4 explained that: *"I can imagine situations where reviewing is straightforward even if the pipeline takes time; however, if the review depends on CI steps, then yes."* Similarly, #P3 noted: *"sometimes the reviewer must wait for the build and reports before starting the review."* These comments suggest that when CR depends on CI outcomes, i.e., the reviewers wait for error-free CI results before starting, CI and CR execution times naturally become correlated.

The more CI pipelines are executed, the more the review time increases. Results for this statement mirrored the previous one: two votes for *"moderately disagree"* and one vote each for *"neutral"* and *"moderately agree"*. Participants' comments further clarified the dynamics behind this correlation. #P3 stated that: *"review may focus only on what the developer marks as complete, regardless of the pipelines."*, suggesting that reviewers may proceed without waiting for CI outcomes. On the other hand, #P2 noted, *"Pipeline delays create uncertainty about reviewing code early, as it might not pass pipeline tests."*, pointing to a more cautious, sequential approach. These responses highlight two distinct scenarios: (i) reviewers conduct code reviews in parallel with CI execution, or (ii) they wait for CI completion before beginning their evaluations. In the first case, CI and CR proceed concurrently; in the second, they are sequential and thus more tightly coupled. While not aiming to determine the better approach, the feedback suggests that sequential workflows strengthen CI–CR correlations, as teams often resolve CI issues before reviewing.

Finding 2: CI and CR execution times tend to be more correlated when the two processes occur sequentially.

4.3 On the Effects Caused by CI Bad Practices on the CR Process (RQ$_3$)

In **RQ$_3$**, we analyze the comments shared by the focus group participants, categorizing them based on the bad CI practices they referenced. These comments provide valuable insights into how specific shortcomings in CI negatively affect the code review process. Out of the fourteen bad practices identified in our previous study [43], participants mentioned ten during the discussion. Table 5 summarizes these practices, ordered by the number of associated reviewer comments. The first column presents an identifier, the second briefly describes each practice, and the third reports the frequency with which each was referenced.

Table 5: Analyzed CI bad practices

ID	CI Bad Practice	# Comments
MP1	Divergent branches	3
MP2	Missing tests in feature branches	2
MP3	Some pipeline tasks are manually triggered	2
MP4	Long build scripts	2
MP5	Developers do not have full control over the environment	1
MP6	Failure notifications are sent only to assigned teams/developers	1
MP7	Lack of testing in a production-like environment	1
MP8	CI server hardware is used for other purposes besides CI execution	1
MP9	External tools are used with default settings	1
MP10	Quality gates are defined without developer input, based only on customer requirements	1
MP11	Feature branches are used instead of feature toggles	0
MP12	Test cases are not organized into folders by purpose	0
MP13	A build fails due to execution instability that should not occur	0
MP14	Developers and operators are maintained as separate roles	0

By analyzing Table 5, we observed that the first four bad practices received the most attention, each with more than one comment from participants. Among them, MP1 (divergent branches) was the most frequently cited. We define divergent branches as feature branches that are outdated from the main branch, increasing the likelihood of merge conflicts and delays. In contrast, practices MP5 to MP10 received only one comment each, and the remaining four were not mentioned at all.

This concentration suggests that the most discussed practices are perceived as more impactful or prevalent in CI environments, possibly because of: (1) *Direct and Immediate Impact*: practices like divergent branches, missing tests, manual steps, and long scripts tend to directly disrupt workflows; (2) *High Frequency of Occurrence*: they may be recurrent in the participants' environments, prompting more detailed concerns; (3) *Visible Consequences*: conflicts, delays, and operational difficulties are readily noticeable; and (4) *Experience and Familiarity*: reviewers may have greater experience dealing with these specific practices.

From the focus group comments, it became clear that CI bad practices can generate problems that directly or indirectly affect the CR process. The full list of comments is available in our replication package. We discuss the most relevant practices as follows.

MP1 - Divergent branches lead to manual conflict resolution, delaying CR. The reviewers emphasized that outdated feature branches often lead to manual conflict resolution, causing significant delays in the code review process. As #P1 noted: *"divergent branches often occur when developers delay updating their branches."* Additionally, #P4 highlighted a lack of experience in handling merges, which exacerbates the problem: *"merges should ideally not generate conflicts."*

MP2 - Missing tests in feature branches, was another frequently mentioned issue. The reviewers stated concerns that the absence of tests increases the reviewers' workload and undermines confidence in code quality. For instance, #P4 stated: *"a pipeline without tests is incomplete."*, while #P2 reinforced the importance of testing by noting: *"the lack of regression testing undermines software quality assurance."* These comments suggest that missing tests not only reduce CI effectiveness but also place additional responsibility on reviewers to verify correctness manually.

Regarding **MP3 - Some pipeline tasks are manually triggered**, the participants expressed frustration with the interruption of automation. Manual intervention was seen as a source of delivery delays and CI inconsistency. For instance, #P3 stated that: *"manual steps cause delays and break CI continuity."* This observation highlights the importance of full automation in maintaining an efficient and predictable integration process. **MP4 - Long build scripts**, were discussed as a source of friction in both development and review workflows. The reviewers emphasized that scripts that are too long or poorly optimized tend to delay CI feedback, creating bottlenecks for subsequent activities. As stated by #P3: *"slow tasks delay the process and can block workflow continuity."*

MP5 - Developers do not have full control over the environment, there was a recognition that restricted access to the build or testing environment can hinder both development and review activities. However, the need for full control was debated. As #P4 stated that: *"developers need some control but not necessarily full control."*, indicating a balance between security, consistency, and developer autonomy.

MP6 - Failure notifications sent only to assigned teams/ developers was also mentioned. Restricted visibility into CI failures was seen as a risk, potentially delaying the identification and resolution of issues that could affect code review. The #P4 remarked: *"notifications should target relevant teams."*, reinforcing the importance of timely and appropriate communication in CI workflows. With respect to **MP7 - Lack of testing in a production-like environment**, concerns were raised about the gap between the test environment and real deployment conditions. This disconnect may allow critical defects to go undetected until the code reaches production or review. As #P4 stated that: *"a production-like environment helps prevent problems from reaching production."*

Finally, **MP9 - External tools used with default settings**, was commented as a subtle but impactful issue. The participants observed that relying on default configurations can limit the effectiveness of tools in the CI pipeline, as stated by #P4: *"default settings often fail to meet the team's needs."* This suggests that customization is essential to align tools with project-specific quality and performance requirements. In summary, from these comments, we concluded that CI bad practices significantly impact CR by causing delays, increasing the review workload, and elevating the risk of undetected issues. Mitigating these practices is essential to ensure a more efficient and reliable review process.

Finding 3: CI bad practices impact the CR process by causing delays, increasing the review workload, and raising the likelihood of undetected problems.

4.4 Benefits and Challenges Generated by CI in CR Process (RQ_4)

As explained in Section 3.1, the final step of the focus group involved a brainstorming session where participants discussed the benefits and challenges that CI introduces to the CR process. We discuss the results in two parts as follows.

1) Benefits. Participants highlighted several ways in which CI enhances the CR effectiveness. Their feedback highlights CI's role in enabling early detection of issues, reducing manual workload, and supporting a more agile and reliable development process. ***Proactive Problem Identification.*** CI tools help reviewers focus their attention by signaling problematic areas in the code. As #P3 noted: *"problem signaling tools make code review easier."* ***Early Bug Detection.*** Frequent and automated testing enables earlier identification of bugs, reducing the likelihood of defects passing unnoticed during CR. As stated by #P2: *"Helps find bugs faster, as tests become more frequent and automated."* ***Efficient Update Distribution.*** Automated deployments speed up and stabilize the release of updates, reducing the risk related to manual procedures. As highlighted by #P2: *"Facilitates the distribution of product updates to the customer."* ***Early Pipeline Issue Detection.*** Identifying CI issues before CR helps prioritize corrections, improving review efficiency. As #P4 explained: *"Some problems can be anticipated during pipeline steps, helping with task prioritization and time management."*

Automated Security Checks. Integrating vulnerability scans into CI prevents critical security issues from reaching the CR phase. The #P4 emphasized that: *"Security and vulnerability checks are essential and can be automated. Such problems would hardly be detected during review."* ***Reduction of Repetitive Tasks.*** Automation minimizes manual effort, allowing reviewers to focus more on qualitative analysis during CR. As stated by #P4: *"Good CI simplifies repetitive tasks through automation."* In summary, these benefits position CI as a key enabler for enhancing CR efficiency and software quality. By anticipating issues, streamlining updates, and minimizing manual work, CI strengthens agile practices and supports continuous improvement in development workflows.

Finding 4: CI adds value to CR by anticipating issues, facilitating update distribution, and minimizing manual work, thereby strengthening the efficiency of the software development lifecycle.

2) Challenges. Despite the benefits, participants also identified several challenges arising from the integration of CI into the CR process. These challenges reflect the limitations of automation, the complexity of real-world systems, and organizational barriers as follows. ***Ensuring Code Quality.*** Even with CI support, maintaining high code quality during CR remains a challenge due to complex issues that are not always captured by automated testing. The #P4 acknowledged, *"ensuring code quality is challenging regardless of the pipeline. A successful pipeline helps, but reviewing itself is a challenge."* ***Conflict Resolution.*** Code conflicts, especially during merges, were cited as a common source of delay, as noted by #P3: *"Code conflicts can significantly delay the process."*

Implementing Complex Tests. Designing tests for complex scenarios, such as those involving databases, demands substantial effort and remains a challenge within CI. In this context, the #P3 stated, *"implementing different types of tests, especially those involving databases in CI, is quite challenging."* ***Process Alignment.*** Misalignment between CI and CR workflows can reduce overall efficiency, highlighting the importance of proper synchronization. As stated by #P3: *"The process affects both CI and review. Aligning the process is challenging."* Finally, ***Risk in Process Adoption.*** Adopting new CI practices was seen as risky, particularly when processes are not yet mature or well understood. As pointed by #P4: *"adapting to any process always involves risks."*

These challenges reflect the complexity of integrating CI and CR effectively. Addressing them requires strategic, flexible approaches to ensure process synchronization, sustain code quality, and foster efficient development cycles.

Finding 5: Combining CI and CR presents challenges, including ensuring code quality, resolving conflicts, and aligning processes, requiring strategic and adaptive approaches to optimize the software development cycle.

5 Study Discussion

The results from RQ_1 revealed a significant association between CI execution time and code review (CR) duration. This finding underscores an interdependence between the two processes, suggesting that CI behavior directly influences how and when developers engage in code review activities.

The correlation emerged both quantitatively, through statistical analysis of 32 metrics, and qualitatively, via developers' insights gathered in focus groups. Notably, CI latency showed strong positive correlations with review time, while CI workload metrics (e.g., total builds and jobs) were also associated with longer review periods. These results are particularly relevant in contexts where CI and CR occur sequentially, as seen in several of the analyzed projects. In such cases, longer CI times delay the start of CR, introducing bottlenecks into the development pipeline.

This interdependence was reinforced by developers' comments, which revealed frustration with long CI queues and the inefficiency of waiting for CI results before initiating reviews. Although parallel execution is theoretically feasible, our findings suggest that in practice, CI often acts as a gatekeeper for CR—whether implicitly or due to formal process constraints. This highlights the importance of reducing CI latency not only to accelerate feedback but also to prevent review delays and developer context switching.

The RQ_3 results further clarify the CI-CR relationship by highlighting how CI bad practices negatively impact the review process. Participants reported issues such as limited test automation, inconsistent builds, and fragile pipelines as harmful to review efficiency and quality. Quantitatively, CI failures were associated with longer review durations and more participants involved, implying increased coordination and effort to resolve problems before approval. These findings are consistent with prior research on CI's role in maintaining development flow and confidence. Our results extend this view by showing that CI issues can propagate beyond integration, directly affecting review timelines and potentially lowering review quality due to reviewer fatigue or distraction.

In contrast, RQ_4 revealed the benefits of well-integrated CI and CR workflows. According to participants, fast feedback, early bug detection, and automation streamline reviews and allow focus on higher-level concerns like design. Moreover, stable CI pipelines reduce cognitive overhead, as reviewers rely on automated checks for syntax and structural correctness.

Nonetheless, some challenges persist. Developers reported difficulties coordinating CI and CR schedules, managing divergent branches, and maintaining review consistency across diverse teams. These challenges suggest that successful integration of CI into CR workflows requires not only appropriate tooling, but also team coordination, process maturity, and cultural alignment. Altogether, the results across all RQs emphasize CI's critical role in shaping CR outcomes. Well-designed CI systems facilitate integration and support efficient reviews, while flawed CI configurations introduce friction and undermine quality. Thus, improving CI performance and stability is essential for teams aiming to streamline reviews and reduce lead time.

Finally, this study contributes to the growing body of research on real-world, closed-source software engineering practices. By combining quantitative and qualitative evidence, we offer actionable insights into how CI and CR interact in practice—and how enhancing one can improve the other.

6 Threats to Validity

We discuss the threats to the validity of this study based on the model proposed by Wohlin et al. [49].

Internal Validity. Variables not considered may influence the relationship between CI and CR processes. To mitigate this, we based our analysis on existing studies and selected a comprehensive set of metrics to capture the nuances between CI and CR. The focus group participants may not fully represent the diversity of development teams. However, given the study context – where code review is performed mainly by project technical leaders – we consider the participant profiles appropriate. Subjectivity may have influenced the interpretation of participants' responses and monitoring data. To address this, the data collection script was validated by an expert, and the focus group application and analysis followed the methodology proposed by Uchôa et al. [47].

External Validity. Findings may not generalize to all organizations due to variations in practices and contexts. Thus, our results should be considered valid for environments similar to the one studied, characterized by CI bad practices, non-advanced development teams, and mature review teams. Changes in CI and CR processes over time could affect result representativeness.

Construct Validity. Different interpretations of CI, CR, and related metrics may impact result comparability. To minimize this, we relied on existing studies, especially Chen et al. [6], and supplemented the metric set based on the study's context. Selected data may not capture all aspects of CI and CR. To mitigate this, we collected metrics from the projects' inception to maximize sampling.

Conclusion Validity. Causal inferences are limited, as results are based on correlations and participant perceptions. However, we consider that the combined quantitative and qualitative analysis provides sufficient support for the conclusions within the industrial context. The study may not cover all nuances of CI and CR processes. Nevertheless, based on a methodology grounded in prior literature and adapted to the project characteristics, we consider the findings valid. A complementary study in an open-source context would further verify the results.

7 Conclusion and Future Work

This study aimed to investigate the intrinsic relationship between CI and CR processes in industrial software development environments. The analysis of the four research questions ($\mathbf{RQ}_1$ to $\mathbf{RQ}_4$) provided a deeper understanding of how these processes interconnect, the impacts of their relationship, and the challenges faced by development teams.

$\mathbf{RQ}_1$ explored the correlation between CI execution time and total code review time, revealing a significant association. The main finding indicates that CI execution time directly influences CR duration, suggesting that improving CI efficiency can positively impact code review. In $\mathbf{RQ}_2$, we found that this interaction is more evident when CI and CR occur sequentially, emphasizing the need for a seamless integration between the processes to optimize development flow and minimize bottlenecks. Regarding the effects of CI bad practices on CR ($\mathbf{RQ}_3$), we observed that CI bad practices can lead to negative outcomes in code review, such as undetected bugs and increased reviewer workload, ultimately affecting code quality. In $\mathbf{RQ}_4$, we identified substantial benefits that CI brings to CR, including proactive problem identification, early bug detection, and efficient update distribution. However, challenges such as ensuring code quality, resolving conflicts, and aligning processes demand adaptive strategies to mitigate their impact.

Overall, the interdependence between CI and CR is complex, involving both technical and human aspects. Careful implementation and integration of these processes are critical to fully realizing the benefits while overcoming the challenges identified. This study provided comprehensive insights into the CI-CR relationship, highlighting the importance of managing them strategically and continuously improving both processes to enhance code quality, accelerate the development cycle, and increase customer satisfaction. As future work, we intend to: (i) apply the study to open-source projects to validate the results obtained, especially about the correlations between CI and RC; (ii) extend the script to collect CR and CI metrics on other platforms; and, (iii) extend the qualitative study of the relationship between CI and CR by capturing the perceptions of both reviewers and developers.

ARTIFACT AVAILABILITY

All artifacts, including datasets, scripts, and documentation from this study are available at https://zenodo.org/records/15807686.

ACKNOWLEDGMENTS

This work was supported by FUNCAP (BP6-00241-00276.01.00/25).

REFERENCES

[1] Cláuvin Almeida, Marcos Kalinowski, Anderson Uchôa, and Bruno Feijó. 2023. Negative effects of gamification in education software. *Information and Software Technology* 156 (2023), 107142. doi:10.1016/j.infsof.2022.107142

[2] A. Bacchelli and C. Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *2013 35th International Conference on Software Engineering (ICSE)*. 712–721. doi:10.1109/ICSE.2013.6606617

[3] João Helis Bernardo, Daniel Alencar da Costa, Uirá Kulesza, and Christoph Treude. 2023. The impact of a continuous integration service on the delivery time of merged pull requests. *Empirical Software Engineering* 28, 4 (2023), 97.

[4] Ekaba Bisong. 2019. *Introduction to Scikit-learn.* Apress, Berkeley, CA, 215–229. doi:10.1007/978-1-4842-4470-8_18

[5] Nathan Cassee, Bogdan Vasilescu, and Alexander Serebrenik. 2020. The Silent Helper. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 423–434. doi:10.1109/SANER48275.2020.9054818

[6] D. Chen, K. T. Stolee, and T. Menzies. 2019. Replication Can Improve Prior Results. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. 179–190. doi:10.1109/ICPC.2019.00037

[7] Adam Debbiche, Mikael Dienér, and Richard Berntsson Svensson. 2014. Challenges When Adopting Continuous Integration. In *Product-Focused Software Process Improvement*, Andreas Jedlitschka, Pasi Kuvaja, Marco Kuhrmann, Tomi Männistö, Jürgen Münch, and Mikko Raatikainen (Eds.). Springer International Publishing, Cham, 17–32.

[8] Wade C Driscoll. 1996. Robustness of the ANOVA and Tukey-Kramer statistical tests. *Computers & Industrial Engineering* 31, 1-2 (1996), 265–268.

[9] PM Duvall. 2018. *Continuous Delivery Patterns and AntiPatterns in the Software LifeCycle.* https://dzone.com/refcardz/continuous-delivery-patterns

[10] Paul M Duvall. 2010. *Continuous Integration.* DZone, Incorporated.

[11] Paul M Duvall, Steve Matyas, and Andrew Glover. 2007. *Continuous integration: improving software quality and reducing risk.* Pearson Education.

[12] Omar Elazhary, Colin Werner, Ze Shi Li, Derek Lowlind, Neil A. Ernst, and Margaret-Anne Storey. 2021. Uncovering the Benefits and Challenges of Continuous Integration Practices. *IEEE Transactions on Software Engineering* (2021). doi:10.1109/TSE.2021.3064953

[13] M. E. Fagan. 1976. Design and code inspections to reduce errors in program development. *IBM Systems Journal* 15, 3 (1976), 182–211. doi:10.1147/sj.153.0182

[14] Nargis Fatima, Suriayati Chuprat, and Sumaira Nazir. 2018. Challenges and Benefits of Modern Code Review-Systematic Literature Review Protocol. In *2018 International Conference on Smart Computing and Electronic Enterprise (ICSCEE)*. 1–5. doi:10.1109/ICSCEE.2018.8538393

[15] Brian Fitzgerald and Klaas-Jan Stol. 2017. Continuous software engineering. *Journal of Systems and Software* 123 (2017), 176–189. doi:10.1016/j.jss.2015.06.063

[16] Martin Fowler. 2006. *Continuous integration.* https://martinfowler.com/articles/continuousIntegration.html

[17] Taher Ahmed Ghaleb, Daniel Alencar Da Costa, and Ying Zou. 2019. An empirical study of the long duration of continuous integration builds. *Empirical Software Engineering* (2019).

[18] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An Exploratory Study of the Pull-Based Software Development Model. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 345–355. doi:10.1145/2568225.2568260

[19] Georgios Gousios and Andy Zaidman. 2014. A Dataset for Pull-Based Development Research. In *Proceedings of the 11th Working Conference on Mining Software Repositories* (Hyderabad, India) *(MSR 2014)*. Association for Computing Machinery, New York, NY, USA, 368–371. doi:10.1145/2597073.2597122

[20] Michael Hilton, Nicholas Nelson, Timothy Tunnell, Darko Marinov, and Danny Dig. 2017. Trade-Offs in Continuous Integration. In *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering* (Paderborn, Germany) *(ESEC/FSE 2017)*. Association for Computing Machinery, New York, NY, USA, 197–207. doi:10.1145/3106237.3106270

[21] Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. 2016. Usage, Costs, and Benefits of Continuous Integration in Open-Source Projects. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering* (Singapore, Singapore) *(ASE 2016)*. Association for Computing Machinery, New York, NY, USA, 426–437. doi:10.1145/2970276.2970358

[22] Ciera Jaspan, Matthew Jorde, Andrea Knight, Caitlin Sadowski, Edward K. Smith, Collin Winter, and Emerson Murphy-Hill. 2018. Advantages and Disadvantages of a Monolithic Repository. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE-SEIP '18)*. Association for Computing Machinery, New York, NY, USA, 225–234. doi:10.1145/3183519.3183550

[23] Jing Jiang, Jiangfeng Lv, Jiateng Zheng, and Li Zhang. 2021. How Developers Modify Pull Requests in Code Review. *IEEE Transactions on Reliability* (2021), 1–15. doi:10.1109/TR.2021.3093159

[24] Gunnar Kudrjavets and Ayushi Rastogi. 2024. Does code review speed matter for practitioners? *Empirical Software Engineering* 29, 1 (2024), 7.

[25] Eero Laukkanen, Juha Itkonen, and Casper Lassenius. 2017. Problems, causes and solutions when adopting continuous delivery—A systematic literature review. *Information and Software Technology* 82 (2017), 55–79. doi:10.1016/j.infsof.2016.10.001

[26] E. Laukkanen, M. Paasivaara, and T. Arvonen. 2015. Stakeholder Perceptions of the Adoption of Continuous Integration – A Case Study. In *2015 Agile Conference*. 11–20. doi:10.1109/Agile.2015.15

[27] Laura MacLeod, Michaela Greiler, Margaret-Anne Storey, Christian Bird, and Jacek Czerwonka. 2018. Code Reviewing in the Trenches. *IEEE Software* 35, 4 (2018), 34–42. doi:10.1109/MS.2017.265100500

[28] Rungroj Maipradit, Dong Wang, Patanamon Thongtanunam, Raula Gaikovina Kula, Yasutaka Kamei, and Shane McIntosh. 2024. Repeated Builds During Code Review: An Empirical Study of the OpenStack Community. In *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering* (Echternach, Luxembourg) *(ASE '23)*. IEEE Press, 153–165. doi:10.1109/ASE56229.2023.00030

[29] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. 2016. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering* 21, 5 (2016), 2146–2189.

[30] Mathias Meyer. 2014. Continuous Integration and Its Tools. *IEEE Software* 31, 3 (2014), 14–16. doi:10.1109/MS.2014.58

[31] Ade Miller. 2008. A hundred days of continuous integration. In *Agile 2008 conference*. IEEE.

[32] R. Morales, S. McIntosh, and F. Khomh. 2015. Do code review practices impact design quality? A case study of the Qt, VTK, and ITK projects. In *Proceedings of the 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. 171–180. doi:10.1109/SANER.2015.7081827

[33] Helena Holmström Olsson, Hiva Alahyari, and Jan Bosch. 2012. Climbing the "Stairway to Heaven" – A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software. In *Proceedings of the 38th Euromicro Conference on Software Engineering and Advanced Applications*. 392–399. doi:10.1109/SEAA.2012.54

[34] Matheus Paixao, Jens Krinke, Donggyun Han, and Mark Harman. 2018. CROP. In *Proceedings of the 15th International Conference on Mining Software Repositories* (Gothenburg, Sweden) *(MSR '18)*. Association for Computing Machinery, New York, NY, USA, 46–49. doi:10.1145/3196398.3196466

[35] Mohammad Masudur Rahman and Chanchal K. Roy. 2017. Impact of Continuous Integration on Code Reviews. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. 499–502. doi:10.1109/MSR.2017.39

[36] Peter C. Rigby and Christian Bird. 2013. Convergent Contemporary Software Peer Review Practices. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering* (Saint Petersburg, Russia) *(ESEC/FSE 2013)*. Association for Computing Machinery, New York, NY, USA, 202–212. doi:10.1145/2491411.2491444

[37] Nishrith Saini and Ricardo Britto. 2021. Using Machine Intelligence to Prioritise Code Review Requests. In *Proceedings of the 43rd International Conference on Software Engineering*. 11–20. doi:10.1109/ICSE-SEIP52600.2021.00010

[38] Jadson Santos, Daniel Alencar da Costa, and Uirá Kulesza. 2024. Holter: Monitoring Continuous Integration Practices. In *Simpósio Brasileiro de Engenharia de Software (SBES)*. SBC, 775–781.

[39] Patrick Schober, Christa Boer, and Lothar A Schwarte. 2018. Correlation coefficients. *Anesthesia & Analgesia* 126, 5 (2018), 1763–1768.

[40] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. 2017. Continuous integration, delivery and deployment. *IEEE Access* (2017).

[41] August Shi, Peiyuan Zhao, and Darko Marinov. 2019. Understanding and Improving Regression Test Selection in Continuous Integration. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. 228–238. doi:10.1109/ISSRE.2019.00031

[42] Dr. D. I. De Silva, W. A. C Pabasara, S. V Sangkavi, Wijerathne L.G.A.T.D, Wijesundara W.M.K.H, and Reezan S.A. 2023. The Effectiveness of Code Reviews on Improving Software Quality: An Empirical Study. 10 pages. doi:10.35940/ijrte.b7666.0712223

[43] Ruben Blenicio Tavares Silva and Carla I. M. Bezerra. 2020. Analyzing Continuous Integration Bad Practices in Closed-Source Projects: An Initial Study. In *Proceedings of the XXXIV Brazilian Symposium on Software Engineering* (Natal, Brazil) *(SBES '20)*. Association for Computing Machinery, New York, NY, USA, 642–647. doi:10.1145/3422392.3422474

[44] Patanamon Thongtanunam, Shane McIntosh, Ahmed E. Hassan, and Hajimu Iida. 2016. Revisiting Code Ownership and Its Relationship with Software Quality in the Scope of Modern Code Review. In *Proceedings of the 38th International Conference on Software Engineering*. Association for Computing Machinery, New York, NY, USA, 1039–1050. doi:10.1145/2884781.2884852

[45] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Influence of Social and Technical Factors for Evaluating Contribution in GitHub. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 356–366. doi:10.1145/2568225.2568315

[46] Alexia Tsotsis. 2011. Meet phabricator, the witty code review tool built inside facebook.

[47] Anderson Uchôa, Caio Barbosa, Willian Oizumi, Publio Blenílio, Rafael Lima, Alessandro Garcia, and Carla Bezerra. 2020. How Does Modern Code Review Impact Software Design Degradation? An In-depth Empirical Study. In *36th*

ICSME.
[48] M. Wessel, A. Serebrenik, I. Wiese, I. Steinmacher, and M. A. Gerosa. 2020. Effects of Adopting Code Review Bots on Pull Requests to OSS Projects. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 1–11. doi:10.1109/ICSME46990.2020.00011
[49] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.
[50] Yue Yu, Huaimin Wang, Vladimir Filkov, Premkumar Devanbu, and Bogdan Vasilescu. 2015. Wait for It. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. 367–371. doi:10.1109/MSR.2015.42
[51] F. Zampetti, G. Bavota, G. Canfora, and M. D. Penta. 2019. A Study on the Interplay between Pull Request Review and Continuous Integration Builds. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 38–48. doi:10.1109/SANER.2019.8667996
[52] Fiorella Zampetti, Simone Scalabrino, Rocco Oliveto, Gerardo Canfora, and Massimiliano Di Penta. 2017. How Open Source Projects Use Static Code Analysis Tools in Continuous Integration Pipelines. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. 334–344. doi:10.1109/MSR.2017.2
[53] Fiorella Zampetti, Carmine Vassallo, Sebastiano Panichella, Gerardo Canfora, Harald Gall, and Massimiliano Di Penta. 2020. An empirical characterization of bad practices in continuous integration. *Empirical Software Engineering* (2020).
[54] Yang Zhang, Gang Yin, Yue Yu, and Huaimin Wang. 2014. Investigating Social Media in GitHub's Pull-Requests. In *Proceedings of the 1st International Workshop on Crowd-Based Software Development Methods and Technologies* (Hong Kong, China) *(CrowdSoft 2014)*. Association for Computing Machinery, New York, NY, USA, 37–41. doi:10.1145/2666539.2666572