UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE QUIXADÁ

PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

MESTRADO ACADÊMICO EM COMPUTAÇÃO

RUBEN BLENICIO TAVARES SILVA

INVESTIGANDO OS EFEITOS DAS MÁS PRÁTICAS DE INTEGRAÇÃO CONTÍNUA EM PROJETOS DE SOFTWARE INDUSTRIAIS

QUIXADÁ

2023

RUBEN BLENICIO TAVARES SILVA

INVESTIGANDO OS EFEITOS DAS MÁS PRÁTICAS DE INTEGRAÇÃO CONTÍNUA EM PROJETOS DE SOFTWARE INDUSTRIAIS

Dissertação apresentada ao Curso de Mestrado Acadêmico em Computação do Programa de Pós-Graduação em Computação do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Ciência da Computação

Orientadora: Prof. Dra. Carla Ilane Moreira Bezerra

QUIXADÁ

2023

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S583i Silva, Ruben Blenicio Tavares.
Investigando os efeitos das más práticas de integração contínua em projetos de software industriais / Ruben Blenicio Tavares Silva. – 2023.
131 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Campus de Quixadá, Programa de Pós-Graduação em Computação, Quixadá, 2023.
Orientação: Profa. Dra. Carla Ilane Moreira Bezerra.

1. Integração Contínua. 2. Más Práticas. 3. Qualidade de Software. 4. Revisão de Código. I. Título.
CDD 005

RUBEN BLENICIO TAVARES SILVA

INVESTIGANDO OS EFEITOS DAS MÁS PRÁTICAS DE INTEGRAÇÃO CONTÍNUA EM PROJETOS DE SOFTWARE INDUSTRIAIS

Dissertação apresentada ao Curso de Mestrado Acadêmico em Computação do Programa de Pós-Graduação em Computação do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Ciência da Computação

Aprovada em:

BANCA EXAMINADORA

Prof. Dra. Carla Ilane Moreira Bezerra (Orientadora)
Universidade Federal do Ceará (UFC)

Prof. Dr. Uirá Kulesza
Universidade Federal do Rio Grande do Norte
(UFRN)

Prof. Dr. Anderson Uchôa
Universidade Federal do Ceará (UFC)

Prof. Dr. Régis Pires Magalhães
Universidade Federal do Ceará (UFC)

À Deus e à minha família, especialmente aos meus pais, Benedito e Consuelo, cujo amor, sacrifício e apoio incansáveis são a base sólida que me sustenta e fortalece.

AGRADECIMENTOS

Agradeço a Deus por ser a fonte de força e inspiração que me ajudou a superar os desafios enfrentados durante esta jornada acadêmica, fornecendo orientação e direção.

Aos meus pais, Benedito e Consuelo, expresso profunda gratidão pelo seu incentivo incondicional e palavras de encorajamento que foram fundamentais em todos os momentos.

À dedicação e apoio dos meus irmãos, Publio e Sueli, especialmente ao meu irmão, cuja colaboração na revisão deste trabalho foi inestimável.

À Profa. Dra. Carla Ilane Moreira Bezerra, minha orientadora, por sua paciência, orientação e sabedoria ao me conduzir nesta dissertação de mestrado.

Ao Prof. Dr. Anderson Gonçalves Uchôa, cuja assistência e orientação foram inestimáveis em vários momentos deste trabalho.

À ilustre banca examinadora, composta pelos estimados Prof. Dr. Uirá Kulesza, Prof. Dr. Regis Pires Magalhães e Prof. Dr. Anderson Gonçalves Uchôa, expresso minha profunda gratidão pela dedicação, tempo e valiosas contribuições concedidas durante a avaliação desta dissertação. Suas observações, críticas construtivas e orientações foram fundamentais para o aprimoramento deste trabalho, enriquecendo-o com *insights* valiosos. Agradeço sinceramente pelo comprometimento e pela generosidade em compartilhar seus conhecimentos, colaborando significativamente para o meu desenvolvimento acadêmico.

Aos estimados colegas do Núcleo de Práticas em Informática, cujo suporte foi inestimável, oferecendo apoio sempre que necessário.

Ao dedicado corpo docente do Programa de Pós-Graduação em Computação da Universidade Federal do Ceará em Quixadá, pelo tempo dedicado e esforços empregados na ministração das disciplinas, contribuindo significativamente para a minha formação neste programa.

Ao Doutorando em Engenharia Elétrica, Ednardo Moreira Rodrigues, e seu assistente, Alan Batista de Oliveira, alunos cuja colaboração na adaptação do *template* deste trabalho foi essencial para atender às normas da biblioteca da Universidade Federal do Ceará (UFC). Agradeço também pela permissão do uso deste *template*, devidamente creditado neste parágrafo.

A todos que, direta ou indiretamente, contribuíram para a conclusão desta dissertação de mestrado, expresso minha sincera gratidão.

"O temor do Senhor é o princípio da ciência;
os loucos desprezam a sabedoria e a instrução."
Provérbios 1:7 ARC – Bíblia Sagrada

(Rei Salomão)

RESUMO

A prática da Integração Contínua (IC) é crucial para a eficiência dos processos de desenvolvimento de software, permitindo a integração frequente de código e a detecção precoce de erros. No entanto, sua má implementação pode resultar em más práticas que comprometem não apenas a eficácia da IC, mas também afetam a qualidade do produto final. Esta pesquisa se concentrou em identificar, compreender e mitigar os efeitos das más práticas de IC em projetos industriais. Ao investigar as más práticas mais comuns, constatamos que problemas como o uso inadequado de *feature branches*, *branches* divergentes e a falta de controle do ambiente pelos desenvolvedores foram recorrentes. Estas más práticas geraram impactos adversos nos projetos industriais, afetando a estabilidade do código, aumentando o tempo e o custo de desenvolvimento, e diminuindo a produtividade das equipes. Quanto à qualidade do software, descobrimos que uma implementação adequada da IC teve efeitos positivos nos atributos internos de qualidade. Houve um aumento geral na coesão do código após a adoção da IC, indicando uma correlação positiva entre uma IC bem executada e uma melhoria na estrutura do código. No entanto, más práticas não resolvidas impactaram negativamente a qualidade do software ao longo do tempo, refletindo em indicadores de qualidade prejudicados e análise estática do código. Analisando a interação entre a IC e a Revisão de Código (RC), identificamos correlações significativas entre métricas de IC e vários aspectos da RC, como tempo de execução e tempo de revisão. Más práticas de IC, como divergências em *branches* e testes inadequados, influenciaram adversamente a RC, aumentando a carga de trabalho dos revisores e atrasando a identificação e correção de problemas no código. Esses resultados sublinham a importância de implementar e manter práticas eficientes de IC para alcançar os benefícios desejados. Mitigar as más práticas de IC não apenas melhora a qualidade do produto final, mas também aumenta a satisfação das equipes de desenvolvimento e otimiza os processos de revisão e correção de código.

Palavras-chave: Integração Contínua. Más Práticas. Qualidade de Software. Revisão de Código.

ABSTRACT

The practice of Continuous Integration (CI) is pivotal for the efficiency of software development processes, enabling frequent code integration and early error detection. However, its improper implementation can result in poor practices that compromise the effectiveness of CI and the final product's quality. This research aimed to identify, understand, and mitigate the effects of CI's bad practices in industrial projects. In investigating the most common bad practices, it was found that issues such as improper use of feature branches, divergent branches, and lack of developer control over the environment were recurrent. These bad practices adversely impacted industrial projects, affecting code stability, increasing development time and cost, and decreasing team productivity. Regarding software quality, it was discovered that proper implementation of CI positively affected internal quality attributes. There was an overall increase in code cohesion following CI adoption, indicating a positive correlation between well-executed CI and an improvement in code structure. However, unresolved bad practices negatively impacted software quality over time, reflecting impaired quality indicators and static code analysis. Analyzing the interaction between CI and Code Review (CR), significant correlations were identified between CI metrics and various aspects of CR, such as execution time and review time. CI bad practices, like branch divergences and inadequate testing, adversely influenced CR, increasing the reviewers' workload and delaying the identification and correction of code issues. These findings underscore the importance of implementing and maintaining efficient CI practices to achieve desired benefits. Mitigating CI's bad practices not only enhances the quality of the final product but also boosts development team satisfaction and streamlines code review and correction processes.

Keywords: Continuous Integration. Bad Practices. Software Quality. Code Review.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

IC	Integração Contínua
RC	Revisão de Código
CI	Continuous Integration
CR	Code Review
RCM	Revisão de Código Moderna
EC	Entrega Contínua

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo, são apresentadas a motivação e a contextualização desta pesquisa, na qual são investigados os efeitos das más práticas de IC dentro de projetos de software. Na Seção 1.1, são discutidas a contextualização e a motivação desta dissertação. Na Seção 1.2, são apresentados os objetivos e as questões de pesquisa. Na Seção 1.3, a metodologia utilizada nesta pesquisa é apresentada. Por fim, na Seção 1.4, é detalhada a estrutura da dissertação, com a organização dos capítulos restantes.

1.1 Motivação e contextualização do problema

IC é um processo que tem como objetivo melhorar a qualidade do software mediante o emprego de diversos procedimentos, como por exemplo: integração frequente do código-fonte em um repositório compartilhado, testes automatizados e análise estática do código (Beller *et al.*, 2017). Os primórdios de IC remetem aos anos 70 (Brooks, 1978). Entretanto, a adoção deste processo na prática só ganhou força nos últimos anos (Felidré *et al.*, 2019).

IC pode gerar alguns benefícios para um projeto de software como, o aumento da frequência de entrega de funcionalidades (Goodman; Elbaz, 2008; Vasilescu *et al.*, 2015), melhoria da produtividade da equipe (Miller, 2008), detecção e resolução precoce dos *bugs* (Beller *et al.*, 2017), e melhoria da comunicação (Downs *et al.*, 2010). Dessa forma, IC tem sido usada amplamente tanto em projetos para fins comerciais quanto para projetos *open-source* (Pinto *et al.*, 2018; Rebouças *et al.*, 2017). De certa forma, os bons resultados relacionados à IC são obtidos graças a ferramentas de *software* que automatizam vários processos de IC, como testes e análise estática do código (Felidré *et al.*, 2019).

Entretanto, existem disparidades entre a teoria e a prática, sendo que em alguns casos os benefícios da IC podem não se manifestar totalmente (Ståhl; Bosch, 2014). Alguns trabalhos recentes relataram a existência de vários obstáculos ligados à implantação do processo de IC nos projetos (Shahin *et al.*, 2017; Ståhl *et al.*, 2017; Vassallo *et al.*, 2018; Vassallo *et al.*, 2019). De modo geral, as maiores dificuldades são relacionadas à efetiva automatização dos testes e à análise dos relatórios de *build* (Zampetti *et al.*, 2020).

Zampetti *et al.* (2020) observaram que mesmo após a implantação do processo de IC, existem algumas más práticas que podem prejudicar o processo como um todo. Felidré *et al.* (2019) também abordam a questão dos *bad smells* relacionados à IC, enfatizando que a utilização

dos processos e ferramentas de IC não necessariamente indica que o projeto ou a equipe irão obter benefícios, pois existem vários aspectos de cultura e/ou organização que resultarão em um pseudoprocesso de IC que os autores chamaram de "teatro da integração contínua". Ghaleb *et al.* (2019) realizaram um estudo para verificar as causas prováveis da longa duração dos *builds* de IC. O estudo identificou que a maioria dos *builds* superam os 10 minutos de duração, sendo que, as maiores causas estão relacionadas a más práticas de IC, tais como: falta de paralelismo nos *builds* e a presença de testes não otimizados e/ou desnecessários.

Espera-se que com a adoção das práticas de IC em um projeto de software a qualidade do mesmo venha a aumentar. Pesquisas recentes sobre IC tem se concentrado em minerar repositórios de código com o objetivo de estudar o impacto da IC na qualidade do software (Elazhary *et al.*, 2021; Saraiva *et al.*, 2023). Em seu estudo Hilton *et al.* (2016), observaram que o uso da automação de *build* produz softwares de maior qualidade. Além disso, vários estudos sobre refatoração de código, por exemplo, investigaram os efeitos desta prática sobre os atributos internos de qualidade do código, tais como: coesão, acoplamento, complexidade, herança e tamanho (Martins *et al.*, 2020; Fernandes *et al.*, 2020; Bibiano *et al.*, 2020).

As más práticas de IC têm sido exploradas por alguns autores como é o caso de Zampetti *et al.* (2020) e Felidré *et al.* (2019). Nestes trabalhos as más práticas e seus efeitos são analisados de forma preliminar sob a perspectiva dos projetos como um todo. No caso de Zampetti *et al.* (2020) os autores se utilizam das experiências dos próprios desenvolvedores com relação ao mau uso da IC nos projetos em que trabalharam. Já o estudo de Felidré *et al.* (2019) se baseia na análise de repositórios *open-source* para localizar indícios de más práticas de IC no ambiente do Travis CI. Estes trabalhos, no entanto, não consideram de modo específico a relação das más práticas de IC com a qualidade dos *softwares* analisados e também não abordam o contexto da industria. São, portanto, necessários mais estudos que forneçam *insights* mais refinados sobre os efeitos das más práticas de IC na qualidade de *software* em projetos industriais.

Através da literatura é possível ver que a IC estabelece um conjunto de procedimentos que fazem com que as equipes de desenvolvimento realizem pequenas alterações em uma base comum, de modo que cada membro tenha uma versão similar do código-fonte em seu ambiente de trabalho (Fitzgerald; Stol, 2017). As modificações precisam de validação, o que pode ser feito automaticamente através de ferramentas de IC que oferecem suporte a configuração de *pipelines*

de IC (e.g., Jenkins[1], Travis CI[2] e GitLab CI[3]). Os *pipelines* de IC são estruturas de controle que mantêm a configuração das etapas necessárias para testar e, se necessário, gerar uma versão do software pronta para executar em um ambiente seja ele de homologação ou de produção (Cano *et al.*, 2021). Muitas organizações normalmente optam por utilizar um arranjo das ferramentas de IC disponíveis que seja compatível tanto com a sua metodologia de trabalho quanto com as necessidades dos seus projetos em desenvolvimento (Shahin *et al.*, 2017; Guerriero *et al.*, 2019).

Além da verificação automática das mudanças promovida pela IC, existe também a verificação manual, mais conhecida como Revisão de Código (RC). RC consiste na verificação manual das mudanças de código pelos membros da equipe de desenvolvimento antes de serem incorporadas na linha principal de desenvolvimento e pode ser implementada de várias maneiras (e.g., listas de discussão, *e-mails* ou de forma oral) (Rigby; Bird, 2013; McIntosh *et al.*, 2014). O objetivo principal da RC é identificar erros e melhorias a serem implementadas no sistema antes que ele entre em produção. Isto porque a correção de erros em uma versão de um sistema que já está em fase de produção tem um custo significativamente maior do que se os mesmos erros fossem corrigidos na fase de desenvolvimento (Baum *et al.*, 2016).

Devido aos benefícios da RC, tanto em projetos industriais quanto de código aberto, os projetos de *software* têm adotado a RC para alavancar seus processos de desenvolvimento (Tsotsis, 2011; Bacchelli; Bird, 2013; Wessel *et al.*, 2020). Atualmente, a RC é também suportada por ferramentas de software, como GitHub[4], GitLab[5] e Gerrit[6], que permitem maior flexibilidade na execução das atividades de RC, sem contar que tem aumentado a necessidade de se trabalhar de forma distribuída (Zampetti *et al.*, 2019). Vale ressaltar, que ferramentas como GitHub e GitLab permitem ainda a integração das práticas de RC e IC através de um recurso chamado solicitação de integração (Rahman; Roy, 2017). As solicitações de integração no GitHub se chamam *pull requests* e no GitLab são chamadas de *merge requests*.

Na literatura, em comparação com estudos que abordam IC e RC individualmente, poucos estudos têm investigado a relação entre estas duas práticas. As pesquisas existentes estão relacionadas de modo geral com: (1) impacto da IC na RC do ponto de vista de fatores como a carga de trabalho dos revisores, o nível de participação dos desenvolvedores na revisão e o tamanho das modificações a serem revisadas (Rahman; Roy, 2017); (2) interação entre as

[1] https://www.jenkins.io/
[2] https://www.travis-ci.com/
[3] https://docs.gitlab.com/ee/ci/
[4] https://github.com/
[5] https://about.gitlab.com/
[6] https://www.gerritcodereview.com/

discussões nas solicitações de integração e o uso de IC dentro das solicitações de integração (Zampetti *et al.*, 2019); (3) percepções e expectativas sobre as mudanças causadas pelo uso de *bots* de revisão de código (Wessel *et al.*, 2020); e, (4) quais as mudanças nas discussões das solicitações de integração após a adoção da IC (Cassee *et al.*, 2020). A maioria das investigações dos estudos de RC e IC utilizam projetos *open-source* (Zampetti *et al.*, 2019; Rahman; Roy, 2017; Wessel *et al.*, 2020; Cassee *et al.*, 2020). Entretanto, existem alguns poucos estudos que exploram a utilização das práticas de RC e IC em projetos industriais (*closed-source*) (Elazhary *et al.*, 2021). Além disso, são necessários mais trabalhos que investiguem o impacto das más práticas de IC no processo de RC.

1.2 Objetivos de pesquisa

Como dito na seção anterior, diversos trabalhos na área de Engenharia de Software apontam que a IC é uma metodologia bastante empregada tanto no contexto de projetos industriais e de código aberto, e que tem trazido bons resultados para estes projetos. No entanto, existe a problemática das más práticas de IC que já foram investigadas em alguns trabalhos que sugerem que estas podem trazer prejuízos para os projetos de *software*. Sendo assim, o objetivo principal do presente trabalho é **investigar o impacto das más práticas de IC no contexto de projetos de software industriais**.

O objetivo principal foi derivado em três questões de pesquisa (QPs), que por sua vez deram origem a três estudos separados onde são analisadas áreas específicas das más práticas de IC. A seguir, são apresentadas e explicadas cada uma das QPs:

QP*$_1$. *Quais as más práticas de IC mais frequentes e com maior impacto negativo em projetos industriais?

Motivação: A primeira questão de pesquisa tem por intuito investigar as más práticas de IC, sobretudo com relação a dois aspectos: frequência de ocorrência das más práticas e o nível de impacto negativo causado por elas. A importância desta questão de pesquisa se caracteriza pela necessidade de verificar (1) se as más práticas de IC apresentadas na literatura ocorrem da mesma forma no ambiente da indústria e, (2) se os impactos destas más práticas são (e o quanto são) percebidos em um ambiente de desenvolvimento de software. Sendo assim, será possível entender melhor como de fato as más práticas de IC ocorrem e como são percebidas pelas equipes de desenvolvimento.

QP*$_2$ *Quais os atributos internos de qualidade mais afetados pelas más práticas de

IC em projetos industriais?

Motivação: A segunda questão de pesquisa possui como foco entender a relação das más práticas de IC com os atributos internos de qualidade de software. A motivação no caso desta questão consiste no fato de que um dos benefícios pregados pela IC é justamente a melhoria na qualidade do software através da detecção e correção dos erros de forma precoce. Dessa forma, pretende-se descobrir se (1) a adoção da IC nos projetos resulta em melhorias nos atributos internos de qualidade e, (2) se (e quais) más práticas de IC podem prejudicar os atributos de qualidade. Com isso, deverá ser possível estabelecer um nível de prioridade para resolução das más práticas de IC de forma a melhorar a qualidade dos softwares analisados pela IC.

QP$_3$ *Como as más práticas de IC afetam a RC em projetos industriais?*

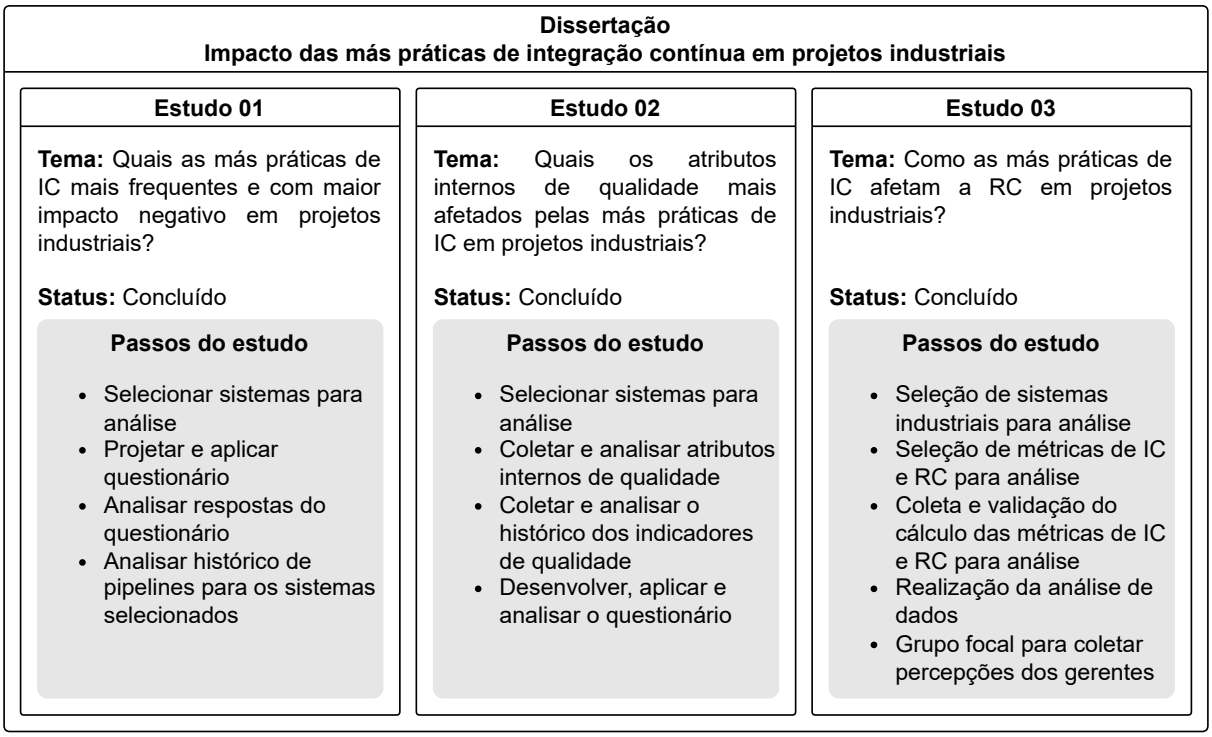
Motivação: Finalmente, na terceira questão de pesquisa o objetivo é examinar como as atividades de RC são afetadas pelas más práticas de IC especificamente no contexto de projetos industriais. A QP_3 surgiu em decorrência da literatura apresentar a RC como um procedimento que visa, assim como a IC, diminuir a quantidade de *bugs* que vão para o ambiente de produção. Além disso, essas duas práticas tem sido bastante empregadas em diversas organizações de desenvolvimento de software, tanto aberto como fechado, o que sugere que bons resultados têm sido alcançados por estas organizações. Assim, busca-se através desta questão avaliar se existe algum nível de correlação entre RC e as más práticas de IC, bem como as percepções dos revisores de código acerca do impacto da utilização de IC juntamente com a RC.

1.3 Metodologia

Para conduzir a presente pesquisa foi feita a opção por utilizar o ambiente de parceiros na indústria de *software* que fizessem uso tanto da IC quanto da RC em seus processos de desenvolvimento. Por isso, a investigação leva em conta apenas o contexto de projetos de código fechado. A Figura 1 apresenta os passos planejados para cada estudo. Como é possível observar, todos os estudos já foram finalizados. O planejamento para a execução dos estudos não teve uma ordem específica para a condução dos mesmos, pois eles não têm relação de precedência. Assim, em um primeiro momento foi realizado o primeiro estudo (relativo à QP_1) e dois estudos foram executados de forma paralela.

Para responder as QPs, foram conduzidos três estudos distintos com o objetivo de investigar de forma específica cada um dos aspectos relacionados às más práticas de IC. A seguir,

Figura 1 – Passos dos estudos que compõem a dissertação



Fonte: elaborado pelo Autor.

são apresentadas as subquestões elaboradas para cada estudo:

- **QP$_1$:** Quais as más práticas de IC mais frequentes e com maior impacto negativo em projetos industriais?
  - *QP$_{1.1}$: Quais são as más práticas de IC mais frequentes nos projetos na perspectiva dos desenvolvedores e gerentes de projetos?*
  - *QP$_{1.2}$: Por quanto tempo as falhas de build permanecem não resolvidas nos projetos?*
  - *QP$_{1.3}$: Quais são os efeitos das más práticas de IC na saúde do software do ponto de vista dos desenvolvedores e gerentes de projeto?*
- **QP$_2$:** Como as más práticas de IC podem afetar a qualidade do software em projetos industriais?
  - *QP$_{2.1}$: Qual é o impacto nos atributos de qualidade interna de sistemas closed-source após a implantação da IC?*
  - *QP$_{2.2}$: O que os indicadores de qualidade ao longo do tempo podem revelar em projetos afetados por más práticas de IC?*
  - *QP$_{2.3}$: Quais más práticas de IC têm a maior prioridade de resolução para a melhoria da qualidade?*
- **QP$_3$:** Como se dá a relação entre os processos de IC e RC em projetos industriais?
  - *QP$_{3.1}$: Quais aspectos do processo de IC e RC estão correlacionados?*

- *$QP_{3.2}$: Quais são as percepções das equipes de desenvolvimento sobre a correlação entre os processos de IC e RC?*
- *$QP_{3.3}$: Como as más práticas de IC afetam a RC?*
- *$QP_{3.4}$: Quais são os benefícios e desafios gerados pela IC no processo de RC?*

Através destas questões de pesquisa, nós visamos mapear os efeitos das más práticas de IC sobre o processo de RC e sobre a qualidade de *software*. Além disso, buscamos capturar as percepções das equipes de desenvolvimento acerca das más práticas de IC no ambiente da indústria.

1.4 Organização

Neste capítulo foram apresentados a motivação, os objetivos, as questões de pesquisa e a metodologia que foi utilizada para esta pesquisa. O restante do documento está organizado da seguinte maneira.

No Capítulo 2, são discutidos os conceitos relacionados aos processos e ferramentas de IC, más práticas de IC, atributos e métricas de qualidade de código, e processos e ferramentas de RC.

O Capítulo 3 enumera as revisões da literatura existentes, bem como os trabalhos relacionados à proposta deste trabalho.

No Capítulo 4, é apresentado um estudo que corresponde à primeira parte desta dissertação, investigando o impacto das más práticas de IC sob a perspectiva das equipes integrantes de 2 projetos industriais.

O Capítulo 5 apresenta o segundo estudo que compõe esta dissertação, no qual nós analisamos o impacto das más práticas de IC sobre a qualidade de software. Neste estudo, foram analisados 9 projetos de nossos parceiros na indústria.

No Capítulo 6, é apresentado o último estudo que compõe esta dissertação, no qual nós analisamos a relação entre os processos de IC e RC no ambiente de nossos parceiros na indústria. Para tanto, consideramos os dados de 10 projetos e as percepções subjetivas de membros dos projetos analisados.

No Capítulo 7 é apresentada a relação do presente trabalho com a *Open Science*. Aqui são enumeradas as contribuições para a comunidade científica e disponibilizados os artefatos gerados no decorrer da pesquisa para fins de reprodutibilidade e extensão.

Por fim, no Capítulo 8, são apresentadas as conclusões finais e a sumarização dos

resultados obtidos nesta dissertação.

2 FUNDAMENTAÇÃO TEÓRICA

A seguir a fundamentação teórica utilizada neste trabalho é apresentada. Foram abordados os conceitos que serão utilizados para alcançar os objetivos deste trabalho, como: processos, ferramentas e más práticas de IC (Seção 2.1), qualidade de código (Seção 2.2) e os processos e ferramentas de RC (Seção 2.3).

2.1 Integração contínua

Booch (1990) apresenta o conceito de IC como um processo onde são criadas versões incrementais de um software. Em seu trabalho Fowler (2006) considera que o intuito da IC é aumentar a taxa de integração de um sistema de modo que os desenvolvedores não permaneçam com suas contribuições por um longo período de tempo em seu ambiente de desenvolvimento. Dessa forma, espera-se que sejam reduzidas as falhas no código e que os erros sejam detectados e eliminados de forma mais efetiva (Hilton *et al.*, 2016).

2.1.1 Processo de integração contínua

Para que a IC funcione de maneira adequada se faz necessária a adoção de algumas práticas nos projetos, tais como:

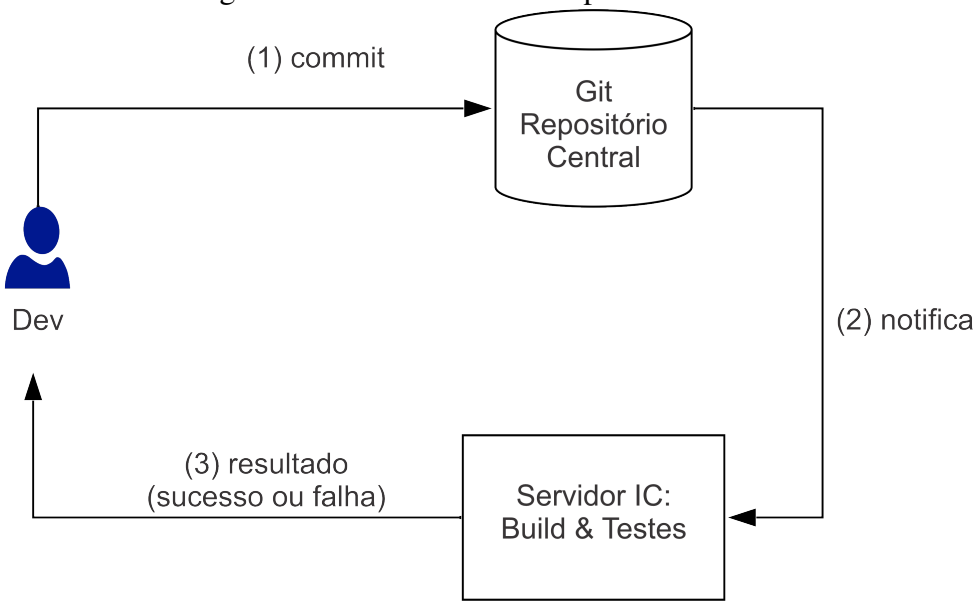
- Manutenção de um repositório único e centralizado do código.
- *Commits* mais frequentes por parte dos desenvolvedores.
- Testes automatizados.
- Uso de ferramentas de automatização para testes e *build* do sistema.
- Uso de ferramentas de análise estática de código.

Em seu trabalho Valente (2020), apresenta um modelo para o processo de IC que pode ser observado na Figura 2. O funcionamento do processo de IC em si é simples: (1) primeiramente o desenvolvedor submete as suas contribuições para o servidor central de um sistema de gerenciamento de versões (no exemplo da Figura tem-se Git[1]), (2) em seguida, o sistema de gerenciamento de versões aciona o servidor de IC que irá realizar as tarefas previamente configuradas (e.g., testes e *build* do código) e (3) por fim, o servidor de IC notifica ao desenvolvedor o resultado da execução do processo de IC, seja ele de sucesso ou falha. Vale ressaltar, que em caso de falha é obrigação do desenvolvedor proceder com a correção do código

[1] https://git-scm.com/

para que o *pipeline* de IC não “quebre”. Sendo assim, o processo itera até que as mudanças sejam aceitas pelo servidor de IC. No contexto do presente trabalho será considerado um processo de IC semelhante ao que é descrito na Figura 2.

Figura 2 – Funcionamento do processo de IC



Fonte: adaptado de Valente (2020)

2.1.2 Ferramentas de integração contínua

Uma das práticas previstas pela IC é o uso de ferramentas para auxiliar o processo. Desta maneira, atividades repetitivas podem ser realizadas de forma automática através de *pipelines* de IC pré-configurados. Diversas ferramentas existentes oferecem suporte às atividades relacionadas a IC. Com isso, é possível configurar um processo no qual as tarefas de compilação, execução de testes, análise estática do código e disponibilização do sistema no ambiente de produção sejam executadas automaticamente. A Tabela 1 apresenta algumas categorias de ferramentas que apoiam o processo de IC, sobretudo com relação a projetos Java e JavaScript. Diferentes combinações destas ferramentas podem ser definidas de forma a atender as necessidades de cada projeto e/ou organização.

Tabela 1 – Ferramentas de IC

CATEGORIA	FERRAMENTAS
Configuração do *pipeline* de IC	GitLab CI; GitHub Actions; Jenkins; CircleCI; Travis CI
Gestão de dependências e compilação	Maven; Gradle; NPM; Yarn
Testes automatizados	JUnity; Jest
Análise estática do código	SonarQube; Codacy

Fonte: elaborado pelo autor.

Neste trabalho, será considerado o seguinte conjunto de ferramentas de IC: (1) o GitLab CI para configuração do *pipeline* de IC, (2) Maven e NPM para prover a gestão de dependências, bem como as tarefas de compilação do código, (3) apenas o JUnit como ferramenta de execução de testes automatizados e (4) o SonarQube para realizar o controle de qualidade através da análise estática do código.

2.1.3 Más práticas de integração contínua

O termo *"bad smell"* foi usado por Fowler (2018) para se referir a indícios de más escolhas no que se refere ao *design* e/ou implementação de sistemas. A mesma expressão foi adaptada por Zampetti *et al.* (2020) para denominar os sintomas de más escolhas relacionadas às práticas de IC. Duvall (2010) alerta para o fato de que a IC precisa ser bem implementada para se obter os seus benefícios e estrutura uma série de más práticas relacionadas ao mal uso de IC.

Alguns trabalhos existentes abordam a problemática das más práticas de IC em projetos de software. Em seu trabalho Duvall (2018), catalogou a partir da literatura um total de 50 padrões e seus respectivos anti-padrões relacionados ao ciclo de vida de *continuous delivery*. Mais recentemente, Zampetti *et al.* (2020) realizaram um estudo empírico para coletar a percepção dos desenvolvedores quanto as más práticas observadas por eles. O resultado de seu trabalho foi um conjunto de 79 más práticas de IC, divididas em 7 categorias, conforme ilustrado na Tabela 2. A seguir, são detalhadas as categorias das más práticas:

- **Repositório:** engloba as más práticas que estão relacionadas a uma organização pobre do repositório de código e mau uso do sistema de controle de versões no que se refere a IC.
- **Infraestrutura:** concentra as más práticas relacionadas a escolhas subótimas de componentes de hardware e software para o *pipeline de* IC.
- **Organização do *build*:** abrange as más práticas relacionadas a má configuração do *pipeline* de IC, como questões ligadas a inicialização do ambiente, as estratégias de limpeza, a decomposição das tarefas, as estratégias de ativação, etc.
- **Manutenibilidade do *build*:** abrange as más práticas que prejudicam a manutenção dos *scripts* ligados ao *pipeline* IC, como ausência de variáveis de ambiente, *scripts* mal comentados, nomes inadequados de variáveis, etc.
- **Garantia da qualidade:** inclui as más práticas associadas a fase de testes e análise estática.
- **Entrega:** está relacionada às más práticas ligadas às atividades de empacotamento e

entrega do *software*.

– **Cultura:** compreende as más práticas que estão ligadas a ações desempenhadas pelos profissionais e a ausência de orientações compartilhadas sobre como os profissionais devem adotar as práticas de IC.

No âmbito deste trabalho será analisada a existência e o impacto em projetos de *software* de todas as más práticas de IC apresentadas na Tabela 2. Além disso, serão investigados os efeitos das más práticas no processo de RC e nos atributos internos de qualidade.

2.2 Qualidade de código

O termo "qualidade" está relacionado a diversas áreas da tecnologia e possui várias definições. Já a qualidade de *software* pode ser definida como a capacidade de adequação às necessidades e expectativas de um sistema (i.e., se o programa implementa uma boa solução e se resolve o problema certo) (Kitchenham, 1989). A qualidade de software visa determinar como um *design* de software está concebido e o quão bom é o grau da estrutura do software, e pode ser medida por diferentes atributos de qualidade (Malhotra; Chug, 2016).

De acordo com (Malhotra; Chug, 2016), atributos de qualidade podem ser classificados em: i) atributos internos de qualidade e ii) atributos externos de qualidade. Atributos externos de qualidade são aqueles que indicam a qualidade do sistema baseado em fatores que não podem ser mensurados somente utilizando artefatos de software. Por exemplo, a manutenibilidade que depende de um conjunto de fatores externos para que seja avaliada, como o tempo de vida do sistema e o ambiente para o qual o sistema está sendo modificado (Dallal, 2013). Atributos internos de qualidade são indicadores chave para medir a qualidade da estrutura do código e assim, atributos internos de qualidade, tais como coesão e acoplamento, podem ser medidos com artefatos de software (Malhotra; Chug, 2016). Quantificar atributos internos de qualidade é mais fácil do que quantificar atributos externos. Por exemplo, o tamanho de uma classe pode ser medida por meio da métrica LOC (Morasca, 2009). Neste trabalho, serão utilizadas métricas para medir os atributos internos de qualidade.

2.2.1 Atributos internos de qualidade

Os atributos internos de qualidade são indicadores que podem ajudar a indicar a qualidade de um *software* (Malhotra, 2019). A literatura apresenta diversos tipos de atributos de

Tabela 2 – Más práticas de IC (Zampetti *et al.*, 2020)

CATEGORIA	METRICA
REPOSITÓRIO	A decomposição do projeto no repositório não segue princípios de modularização
	Os casos de teste não são organizados em pastas com base em seus propósitos
	O espaço de trabalho local e remoto não estão alinhados
	Número de branches não se encaixa nas necessidades/características do projeto
	Falta uma branch de liberação estável
	É utilizada a estratégia de desenvolvimento "feature branches"em vez de "feature toggles"
	Branches divergentes (ausência de merge/commit diário)
	Os artefatos gerados pelo pipeline de CI são versionados
	Blobs(libs, imagens docker, dependências) são desnecessariamente verificados em todas os builds ao invés de serem armazenados em cache
	Recursos relacionados a pipelines não são versionados
INFRAESTRUTURA	Recursos relacionados a um mesmo pipeline são distribuídos em vários servidores
	O hardware do servidor de CI é utilizado para outros fins que não a execução do framework CI
	Ferramentas externas são utilizadas com suas configurações padrão
	Diferentes versões de ferramentas/plugins são instaladas no mesmo servidor
	Plugins diferentes são usados para realizar a mesma tarefa no mesmo processo de build
	Uma tarefa é implementada utilizando uma ferramenta/plugin inapropriada
	Uso de scripts shell para uma tarefa para a qual existe um plugin adequado
ORGANIZAÇÃO DO BUILD	Estratégia de limpeza do ambiente de build inapropriada
	Ausência de sistema de gerenciamento de pacotes(dependências)
	São utilizados jobs de build grandes e incoerentes
	Os builds monolíticos são utilizados no pipeline
	Os jobs de build independentes não são executados em paralelo
	Apenas o último commit é construído(buildado), abortando builds obsoletos na fila de execução
	As etapas de build não são devidamente ordenadas
	Passos/estágios do pipeline são pulados arbitrariamente
	As tarefas não são adequadamente distribuídas entre os diferentes estágios de build
	Builds incrementais são utilizados sem nunca "buildar"o projeto inteiro a partir do zero
	Estratégia de acionamento do build deficiente
	Builds privados(no ambiente do desenvolvedor) não são usados
	Algumas tarefas do pipeline são iniciadas manualmente
	Uso de builds noturnos(agendados)
	Projetos inativos estão sendo analisados
	Um build é bem sucedido quando uma tarefa falha ou um erro é lançado
	Um build falha por causa de alguma falha na execução, sendo que não deveria
	O gerenciamento de dependências não é utilizado
	Inclusão de dependências desnecessárias
	Algumas tarefas são executadas sem reportar claramente os seus resultados na saída do build
	As saídas de diferentes tarefas são misturadas na saída do build
	As notificações de falhas somente são enviadas para os times ou desenvolvedores que estão explicitamente inscritos para a tarefa
	Ausência de mecanismo de notificação
	Os relatórios de build contém informações verbosas e irrelevantes
	O timeout não está adequadamente configurado
	Tarefas desnecessárias são alocadas no processo de build
	Tempo de build para o estágio de commit supera a regra dos 10 minutos
	Passos de rebuild desnecessários são realizados
	Os dados de autenticação são armazenados em código-fonte no sistema de controle de versões
MANUTENIBILIDADE DO BUILD	Caminhos absolutos ou dependentes da máquina são usados
	Os scripts de build são altamente dependentes da IDE
	As variáveis de ambiente não são totalmente utilizadas
	As configurações de build são clonadas em diferentes ambientes
	Os jobs de build não são parametrizados
	Scripts de build demorados
	Ausência de "smoke test", conjunto de testes para verificar a testabilidade do build
	Convenção rigorosa de nomenclatura ausente ou pobre para jobs de build
GARANTIA DA QUALIDADE	Testes não são realizados em um ambiente semelhante ao de produção
	As ferramentas de cobertura de código não são executadas durante a realização dos testes de unidade e integração
	Os thresholds de cobertura são fixados com base no que foi alcançado em builds anteriores
	Os thresholds estão muito elevados
	Ausência de testes nas branches de atividade
	Todas as permutações de funcionalidades alternadas são testadas (muitos testes irrelevantes)
	Os recursos de produção são usados para propósito de testes
	Os testes não estão totalmente automatizados levando a um build não reproduzível
	A suíte de testes contém flaky tests(resultam em sucesso ou erro de forma arbitrária)
	Má escolha no subconjunto de casos de teste para executar no servidor CI
	Testes defeituosos são reexecutados no mesmo build
	Quality gates são definidos sem desenvolvedores, considerando apenas o que é ditado pelo cliente
	Uso de quality gates para monitorar a atividade de desenvolvedores específicos
	Verificações desnecessárias da análise estática são incluídas no processo de build
ENTREGA	Artefatos gerados localmente são implantados
	Ausência de repositório de artefatos
	Ausência de estratégia de rollback
	Estratégia de nomeação de release ausente
	Ausência de verificação para os entregáveis
CULTURA	As modificações são submetidas antes de se consertar um build defeituoso anterior
	A reunião/discussão da equipe é realizada pouco antes da submissão para a branch principal
	Desenvolvedores e operadores são mantidos em papéis separados(ausência de DevOps)
	Os desenvolvedores não possuem um controle completo do ambiente
	As falhas de build não são imediatamente resolvidas, dando prioridade para outras mudanças
	As notificações de tarefas são ignoradas

Fonte: elaborado pelo autor.

qualidade. No contexto deste trabalho, serão analisados apenas os seguintes atributos internos de qualidade:

– **Acoplamento**: o acoplamento é o grau de interdependência entre classes ou módulos

(Chidamber; Kemerer, 1994). Um sistema com alto grau de acoplamento pode ser difícil de manter e reutilizar.

- **Coesão**: a coesão se refere ao quão os elementos internos de um módulo estão relacionados entre si. Em caso de baixa coesão há uma alta probabilidade de ocorrência de *bugs* e de o módulo ficar demasiadamente complexo.
- **Complexidade**: este atributo se refere ao grau de sobrecarga de responsabilidades e tomada de decisão em um módulo. Um código muito complexo provavelmente será também ilegível.
- **Herança**: a herança representa as relações entre superclasses e subclasses. Através da herança é possível promover a reusabilidade dentro do *software*. Isto, pelo menos à princípio é bom, entretanto grandes hierarquias em um software podem ocasionar problemas de manutenção.
- **Tamanho**: pode-se dizer que este é o mais simples dos atributos internos de qualidade. Aqui é medido o comprimento de um módulo. É perfeitamente normal o tamanho dos componentes crescer com o andamento do desenvolvimento de um projeto de *software*, entretanto módulos muito grandes também poder resultar em problemas de manutenção.

Embora a análise apenas destes 5 atributos de qualidade possa não ser suficiente para inferir com precisão a qualidade de um software, eles fornecem uma base para uma investigação inicial. Neste trabalho é examinado o efeito das más práticas de IC em relação a estes 5 atributos internos de qualidade.

2.2.2 Métricas de qualidade de código

Para cada atributo interno de qualidade existe um conjunto de métricas que permitem mensurar o grau de qualidade de um sistema através de valores reais. Diversos trabalhos se utilizam das métricas de qualidade para analisar atributos específicos em seus sistemas e provêm recomendações de métricas de qualidade adequadas para diversos contextos de aplicação (Hamdi *et al.*, 2021; Mladenova, 2020; Dalla Palma *et al.*, 2020). Mens e Tourwe (2004) também fazem uso das métricas de qualidade de forma a medir a saúde dos *softwares* analisados. De modo geral, as métricas de qualidade servem para indicar características específicas dos sistemas analisados. Com isso, torna-se mais fácil realizar análises (e.g., se determinada metodologia ou processo traz melhorias para a saúde de um *software*), planejar e executar melhorias em um sistema.

Existem diversas ferramentas que dão suporte à captura e análise das métricas de

qualidade. No contexto deste trabalho foi selecionada a ferramenta Understand[2] para proceder com a coleta dos dados necessários relativos às métricas de qualidade de *software*. A Tabela 3 apresenta um conjunto de 13 métricas que foram utilizadas no presente trabalho. Além disso, as métricas são classificadas de acordo com os atributos de qualidade e é apresentada uma descrição relativa ao valor de cada uma das métricas.

Tabela 3 – Métricas dos atributos internos de qualidade

ATRIBUTOS	MÉTRICA	DESCRIÇÃO
Coesão	Falta de coesão dos métodos (LCOM2) (Chidamber; Kemerer, 1994)	Mede a coesão de uma classe. Quanto maior o valor dessa métrica, menos coesiva é a classe.
Acoplamento	Acoplamento entre objetos (CBO) (Chidamber; Kemerer, 1994)	Número de classes a que uma classe está acoplada. Quanto maior o valor dessa métrica, mais acoplados são as classes e métodos.
Complexidade	Média de complexidade ciclomática (ACC) (McCabe, 1976)	Média de complexidade ciclomática de todos os métodos aninhados. Quanto maior o valor dessa métrica, mais complexas são as classes e os métodos.
	Soma de complexidade ciclomática (SCC) (McCabe, 1976)	Soma da complexidade ciclomática de todos os métodos aninhados. Quanto maior o valor dessa métrica, mais complexas são as classes e os métodos.
	Aninhamento (MaxNest) (Lorenz; Kidd, 1994)	Nível máximo de aninhamento das estruturas de controle. Quanto maior o valor desta métrica, mais complexas são as classes e métodos.
	Complexidade essencial (EVG) (McCabe, 1976)	Medida do grau em que um módulo contém estruturas desorganizadas. Quanto maior o valor dessa métrica, mais complexas são as classes e métodos.
Herança	Número de filhos (NOC) (Chidamber; Kemerer, 1994)	Número de subclasses de uma classe. Quanto maior o valor dessa métrica, maior é o grau de herança de um sistema.
	Profundidade da árvore de herança (DIT) (Chidamber; Kemerer, 1994)	O número de níveis que uma subclasse herda dos métodos e atributos de uma superclasse na árvore da herança. Quanto maior o valor dessa métrica, maior é o grau de herança de um sistema.
	Classes base (IFANIN) (Destefanis *et al.*, 2014)	Número imediato de classes base. Quanto maior o valor dessa métrica, maior é o grau de herança de um sistema.
Tamanho	Linhas de código (LOC) (Lorenz; Kidd, 1994)	Número de linhas de código excluindo espaços e comentários. Quanto maior o valor dessa métrica, maior o tamanho do sistema.
	Linhas com comentários (CLOC) (Lorenz; Kidd, 1994)	Número de linhas com comentários. Quanto maior o valor desta métrica, maior o tamanho do sistema.
	Classes (CDL) (Lorenz; Kidd, 1994)	Número de classes. Quanto maior o valor dessa métrica, maior será o tamanho do sistema.
	Métodos de instância (NIM) (Lorenz; Kidd, 1994)	Número de métodos de instância. Quanto maior o valor dessa métrica, maior o tamanho do sistema.

Fonte: Elaborado pelo autor.

2.3 Revisão de código

A RC consiste em uma inspeção manual do código-fonte por integrantes da equipe de desenvolvimento (diferente do autor do código) e é reconhecida como uma ferramenta valiosa para melhorar a qualidade dos projetos de software (Ackerman *et al.*, 1984; Ackerman *et al.*, 1989). Fagan (1976) estruturou um processo formal para a RC chamado de "inspeção de código". Ao longo do tempo, a RC tem sido avaliada em diversos trabalhos que evidenciam os benefícios da inspeção de código, sobretudo com relação à detecção de *bugs* no código. Entretanto, por ser um processo complexo a sua aplicação prática não obteve uma grande adesão pela industria (Shull; Seaman, 2008).

Mais recentemente, muitas organizações passaram a adotar processos de RC mais enxutos visando compensar a ineficiência trazida pela inspeção de código clássica (Rigby; Bird, 2013). Estes processos de RC mais resumidos podem ser incluídos no contexto da Revisão de

[2] https://www.scitools.com/

Código Moderna (RCM). Esta metodologia é caracterizada por (1) poder se aplicada de modo informal, (2) ser apoiada por ferramentas de software, (3) ter características assíncronas e (4) focar especificamente na revisão das mudanças no código. No contexto desta dissertação, ao se falar de RC, será considerado o conceito da RCM, isto devido ao fato de que os parceiros da industria abordados no estudo implementam a RC nos moldes da RCM. A seguir, são apresentadas as definições gerais relacionadas ao processo bem como às métricas de RC existentes.

2.3.1 Processo de revisão de código

Existem diversos processos de RC desenvolvidos normalmente de acordo com as necessidades de cada projeto/organização. Entretanto, neste trabalho serão apresentados três processos de alto nível para a execução da RC.

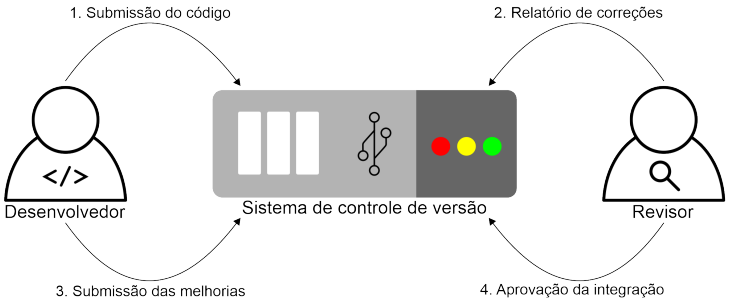
Inspeções de código. As inspeções de código constituem o primeiro método definido formalmente para realizar a RC. Trata-se de um processo altamente estruturado que envolve planejamento, análise, preparação, reunião para inspeção, retrabalho e acompanhamento das mudanças (Fagan, 1976). Vale ressaltar, que as inspeções de código por princípio são realizadas de forma síncrona com a presença dos revisores e dos responsáveis pelas mudanças em análise.

Revisão assíncrona via *e-mail*. Principalmente em projetos *open-source*, o processo de RC é realizado de forma assíncrona via *e-mail* e tem sido bastante empregado devido à praticidade na forma de comunicação dos revisores e desenvolvedores. Da mesma forma que as inspeções, a revisão via *e-mail* tem o objetivo de identificar defeitos no software (Rigby *et al.*, 2014), contudo o processo é diferente. Primeiramente os desenvolvedores submetem um conjunto de mudanças para uma lista de *e-mail* do projeto. Em seguida, os revisores avaliam as mudanças e solicitam correções/modificações. Por fim, quando o conjunto de mudanças adquire um alto nível de qualidade (definido pela equipe do projeto) ele é integrado ao código.

Revisão apoiada por ferramentas. O processo de revisão apoiado por ferramentas surgiu com a necessidade de estruturar a logística da RC. As ferramentas que suportam o processo de RC normalmente funcionam da seguinte forma: (1) o desenvolvedor submete um conjunto de mudanças para revisão, (2) os revisores obtêm acesso ao conjunto de mudanças e podem comparar com o código atual, (3) incia-se uma ou várias discussões em pontos específicos do conjunto de mudanças com o autor e os revisores e por fim (4) o autor sugere um conjunto de alterações no conjunto de mudanças original de forma que o código seja integrado à linha principal de desenvolvimento. A Figura 3 apresenta o processo de RC apoiado por ferramentas.

No contexto deste trabalho será considerado apenas o modelo de RC apoiado por ferramentas.

Figura 3 – Processo de revisão de código apoiado por ferramentas



Fonte: elaborado pelo Autor.

2.3.2 Métricas de revisão de código

Existem diversos aspectos que podem ser analisados em relação à RC. Nesse contexto, com o intuito de promover uma análise mais sólida do processo de RC em projetos de software, diversos estudos definem métricas de RC que auxiliam no entendimento dos fatores da RC na prática (Gousios; Zaidman, 2014; Gousios *et al.*, 2014; Tsay *et al.*, 2014; Yu *et al.*, 2015; Zhang *et al.*, 2014; Cassee *et al.*, 2020; Chen *et al.*, 2019). Sendo assim, as métricas de RC constituem-se uma poderosa ferramenta, sobretudo em estudos empíricos onde há a análise de impacto de determinada prática/processo/metodologia na RC e vice-versa. No presente trabalho, as métricas de RC serão utilizadas com o objetivo de analisar os efeitos da IC na RC. A Tabela 4 apresenta uma série de métricas de RC juntamente com a referência para trabalhos onde elas aparecem.

Tabela 4 – Métricas de RC

MÉTRICA	Gousios e Zaidman (2014)	Gousios *et al.* (2014)	Tsay *et al.* (2014)	Yu *et al.* (2015)	Zhang *et al.* (2014)	Cassee *et al.* (2020)	Chen *et al.* (2019)
Tempo de merge	■						
Tempo de fechamento	■						
Tempo de revisão do código							
Número de participantes	■	■					
Comentários de revisão						■	■
Comentários gerais						■	■
Comentários efetivos						■	■
Total de comentários	■	■	■	■			
Tempo até a primeira resposta				■			■
Número de menções					■		
Número de commits	■	■	■	■			■
Mudanças de código-fonte	■	■		■			■
Mudanças em código de testes	■	■					

Fonte: elaborado pelo autor.

Nos sistemas de controle de versão existentes, as métricas apresentadas na Tabela 4 normalmente estão associadas às solicitações de integração. Estas solicitações de integração

são caracterizadas como *pull requests* no ambiente do Github e do BitBucket[3] e como *merge requests* no contexto do GitLab. Para possibilitar a coleta das métricas de RC, neste trabalho será implementado um *script* que fará o acesso à API do sistema de controle de versão (neste caso, o GitLab) e computará as métricas gerando, por fim, um relatório que poderá ser utilizado para as analises.

2.4 Conclusão

Este capítulo apresentou os principais conceitos que serão utilizados por este trabalho. O objetivo deste trabalho é investigar os efeitos das mas práticas de IC em projetos de *software open-source*. Sendo assim, para apoiar a pesquisa foram apresentados os conceitos relacionados (1) ao processo de IC, (2) ferramentas de IC, (3) más práticas de IC, (4) atributos e métricas de qualidade de *software* e (5) o processo e métricas de RC.

[3] https://bitbucket.org/product/

3 TRABALHOS RELACIONADOS

Este capítulo apresenta os principais trabalhos relacionados a esta pesquisa. Foram agrupados os trabalhos relacionados em quatro tópicos: (1) revisões da literatura sobre integração contínua (Seção 3.1); (2) más e boas práticas de integração contínua (Seção 3.2); (3) integração contínua e qualidade de *software* (Seção 3.3); e, (4) integração contínua e revisão de código (Seção 3.4). Na Seção 3.5 é realizada uma comparação dos trabalhos relacionados com a proposta do presente trabalho. Finalmente, na Seção 3.6 são apresentadas as conclusões do capítulo.

3.1 Revisões da literatura sobre integração contínua

Em geral, poucas investigações foram implementadas com o intuito de analisar o estado da arte no que se refere à IC. A seguir, são apresentados dois trabalhos que se propõem a identificar os trabalhos existentes que abordam a temática da IC em projetos de *software*.

Shahin *et al.* (2017) realizaram uma revisão sistemática da literatura com o objetivo de sintetizar as abordagens, ferramentas, desafios e práticas existentes relacionadas à adoção e implementação das práticas contínuas, que incluem a IC. Os autores analisaram trabalhos publicados entre 2004 e junho de 2016 e selecionaram 69 artigos. Como resultado, foram identificadas 30 abordagens/ferramentas que visam facilitar a adoção das práticas de IC através: (1) da redução do tempo de *build* e dos testes na IC; (2) melhoria da visibilidade e compreensão dos resultados de *build* e teste de IC; (3) suporte a testes automáticos e semiautomáticos; (4) detecção de falhas de IC; (5) melhoria na segurança e escalabilidade dos *pipelines* de IC; e (6) melhoria na confiabilidade do processo de IC. No trabalho também foram identificados fatores primordiais para o sucesso da IC em projetos de software, que são:

- Testes automatizados.
- Conscientização da equipe e transparência do processo.
- Bons princípios de projeto.
- Relacionamento com o cliente.
- Qualificação e motivação da equipe.
- Conhecimento do domínio da aplicação e da infraestrutura adequada.

Em seu trabalho, Soares *et al.* (2022) buscaram através de uma revisão sistemática da literatura, identificar como a IC impacta o desenvolvimento de software. A partir de um conjunto inicial de 479 estudos, os autores selecionaram um total de 101 trabalhos que avaliam

de modo empírico o uso da IC com relação a diversas áreas do desenvolvimento de *software*. Sua metodologia de pesquisa se baseou primeiramente na extração dados referentes (1) ao ambiente de IC, (2) efeitos da IC e (3) metodologia de IC empregada, e por fim foi aplicado o método da síntese temática para obter as conclusões. Como resultados, o estudo descobriu que as pesquisas existentes ressaltam a existência de efeitos positivos da IC como uma melhoria na cooperação das equipes de desenvolvimento. Entretanto, existem também efeitos negativos como a adição de maior complexidade técnica e de processos. Os autores também chegaram a um total de 31 afirmações extraídas a partir dos trabalhos analisados, acerca dos impactos positivos e negativos da IC em projetos de software. As afirmações foram divididas nas seguintes categorias: (1) atividades de desenvolvimento, (2) processo de *software*, (3) garantia da qualidade, (4) modelos de integração, (5) problemas e falhas e (6) padrões de *build*.

Como é possível observar, ainda existem lacunas de pesquisa a serem exploradas, principalmente na temática da proposta do presente trabalho de dissertação: o impacto das más praticas de IC em projetos de *software* industriais. As próximas três Seções apresentam os trabalhos relacionados à proposta desta dissertação.

3.2 Más e boas práticas de integração contínua

Felidré *et al.* (2019) realizaram um estudo para investigar um conjunto de más práticas de IC em 1270 projetos. Essas práticas inadequadas estão relacionadas ao uso do IC (1) com *commits* pouco frequentes na *branch* principal, (2) em projetos de software com baixa cobertura de teste, (3) com *builds* que permanecem quebrados por longos períodos de tempo e (4) com tempo de *build* consideravelmente longo. Como resultados, os autores identificaram que embora algumas práticas ruins sejam comumente empregadas, como *commits* pouco frequentes na *branch* principal (em ± 60% dos projetos), outras práticas ruins não são tão frequentes (como um *build* demorando muito para ser processado). Ainda observou-se que, 85% dos projetos estudados têm pelo menos um *build* quebrado que leva mais de quatro dias para ser consertado.

Zampetti *et al.* (2020) conduziram uma investigação utilizando entrevistas semiestruturadas com 13 especialistas e minerando mais de 2.300 postagens do *Stack Overflow*. Como resultado, os autores obtiveram um catálogo de 79 *bad smells* de IC pertencentes a 7 categorias relacionadas a diferentes dimensões de um processo de gerenciamento de *pipeline* de IC. A relevância das descobertas foi validada através da consulta a 26 desenvolvedores da indústria e pôde-se observar que apesar dos resultados de algumas más práticas encontradas confirmadas

em estudos anteriores na literatura (mau uso das *branches*), outras más práticas contradizem os achados da literatura (como, *builds* noturnos). No presente trabalho, será utilizado o catálogo de más práticas de IC elaborado por Zampetti *et al.* (2020), investigando os *bad smells* de IC mais frequentes e os efeitos dessas más práticas na saúde do software.

Vassallo *et al.* (2019) realizaram um estudo com 124 desenvolvedores sobre os anti-padrões de IC. O seu resultado foi a implementação de uma ferramenta capaz de realizar a análise de forma automática dos sistemas com o objetivo de identificar as más práticas de IC. Com o auxílio da ferramenta foi executada a análise de 18.474 *logs* de *build* referentes a 36 projetos JAVA de código aberto. Esta verificação reportou um total de 3.823 avisos espalhados nos sistemas. O relatório foi ainda validado com o auxílio de uma pesquisa envolvendo o *feedback* 13 desenvolvedores dos projetos e o *feedback* geral de mais 42 desenvolvedores.

Gallaba e McIntosh (2020) desenvolveram uma pesquisa para investigar o mal uso dos recursos fornecidos pelos serviços de IC. Foram utilizados no estudo 9.312 sistemas de código aberto. Em uma análise inicial dos recursos, foi detectado que a maior parte do código de IC está relacionado à configuração dos nós de processamento de trabalho e não à implantação das funcionalidades desenvolvidas. Para averiguar o mal uso dos recursos, foi implementada uma ferramenta capaz de identificar quatro anti-padrões pré-definidos. Os autores implementaram uma ferramenta para remoção de anti-padrões, que remove de forma eficaz 69,60% dos anti-padrões mais frequentes.

Mohammad (2016) discute a importância da integração contínua e da automação no processo de desenvolvimento de software. O artigo apresenta uma visão geral da integração contínua e descreve como ela pode ser implementada em um ambiente de desenvolvimento de software. O autor também discute a importância da automação no processo de desenvolvimento de software e como ela pode ser usada para melhorar a eficiência e a qualidade do software. O artigo conclui que a integração contínua e a automação são fundamentais para o sucesso do processo de desenvolvimento de software e devem ser implementadas em todas as etapas do processo.

Os estudos apresentados nesta seção abordam o mau uso dos recursos e as más práticas de IC existentes. No presente trabalho as más práticas de IC catalogadas serão utilizadas com o objetivo de mapear os seus efeitos (1) na qualidade de *software* e no processo de RC. Estes dois fatores não foram abordados de forma específica nos trabalhos existentes. Além disso, assim como os estudos apresentados, este trabalho é de caráter empírico, contudo o contexto de

pesquisa é voltado para o ambiente da indústria e não de projetos *open-source*.

3.3 Integração contínua e qualidade de software

Hamdan e Alramouni (2015) realizaram um estudo empírico a fim de investigar o impacto da IC sobre a qualidade do processo de desenvolvimento de *software*. Eles aplicaram um estudo de caso com base no projeto de um sistema financeiro que fornecia serviços de pagamento. Os autores analisaram o estado anterior e posterior da implantação das práticas de IC no projeto utilizando vários atributos de qualidade relacionados ao ciclo de vida de desenvolvimento. Como resultado, o estudo revelou que após a implantação da IC os desenvolvedores levaram menos tempo para desenvolver novas *features* e para consertar defeitos no *software*. Além disso, foram introduzidas melhorias em diversas áreas do processo de desenvolvimento como documentação e testes.

Alizadeh *et al.* (2019) tratam das práticas de refatoração de código integradas ao ambiente de IC que constituem uma importante ferramenta para a melhoria da qualidade do *software*. A problemática citada no estudo se baseia no fato de que o processo manual de refatoração é caro e as ferramentas de refatoração existentes em sua maioria não permitem uma fácil integração dentro de um *pipeline* de IC. Sendo assim, os autores propõem o RefBot, um software inteligente integrado ao sistema de controle de versão e ao *pipeline* de IC, que é responsável por monitorar continuamente a qualidade do código e propor refatorações, caso necessário. O *bot* foi submetido à avaliação tanto quantitativa quanto qualitativa por desenvolvedores da indústria e os resultados sugerem que a utilização da ferramenta integrada ao ambiente de IC pode encorajar os desenvolvedores a aplicar refatorações de código.

Em seu trabalho, Freitas (2020) investigou a relação entre a utilização dos processos de IC e a qualidade do código. O autor analisou as correlações entre métricas de IC, qualidade de código (especificamente no contexto de sistemas orientados a objetos) e de projeto. Como resultado principal, não foram encontrados indícios suficientes para indicar que a utilização dos processos de IC interfere diretamente na qualidade de código de projetos de *software*. Além disso, foram feitas descobertas relevantes, tais como: (1) projetos com processo de *build* mais rápido tendem a ser melhores estruturados (a nível de classes e pacotes), porém são mais suscetíveis a apresentar alto acoplamento; (2) a priorização das correções de erros de *build* podem fazer com que o sistema tenha uma menor árvore de herança, bem como classes melhor estruturadas; e, (3) uma maior cobertura de testes faz com que as operações individuais sejam menos complexas

mas prejudica a organização das operações a nível de sistema.

Elazhary *et al.* (2021) realizaram um estudo em três empresas de desenvolvimento de software. Para este estudo, foi realizada uma análise dos dados registrados das atividades desenvolvidas, com o objetivo de identificar o uso de *builds* automatizados e a frequência dos *commits*. Uma pesquisa voltada para os desenvolvedores foi desenvolvida para identificar o uso, benefícios e outros efeitos que a implementação da IC pode gerar. Os autores descobriram que: (1) apesar do uso da IC em projetos de *software*, a forma como ela é aplicada e as ferramentas utilizadas podem alterar a percepção sobre os benefícios percebidos em relação à qualidade; (2) IC pode criar restrições no processo de desenvolvimento que acabam por prejudicar o *feedback*; e, (3) principalmente para os pesquisadores, é importante considerar o contexto do projeto para avaliar o uso da IC.

Lai *et al.* (2021) abordam a problemática da manutenção de softwares críticos, especificamente de sistemas bancários. O objetivo principal do trabalho é melhorar a qualidade e a eficiência do processo de manutenção de software, uma vez que problemas nesta fase do ciclo de vida de desenvolvimento podem ocasionar severas perdas. Dessa forma, os autores realizaram uma redefinição do fluxo de trabalho de manutenção de *software* utilizando como base as práticas de IC para promover a automatização do processo. Como resultado, o estudo apresenta o modelo CPQM que tem como objetivo identificar falhas no processo e auxiliar na garantia da qualidade de *software*.

Em seu trabalho Saidani *et al.* (2021) investigam o impacto da Integração Contínua (CI) na prática de refatoração de software. O estudo foi realizado no contexto do TravisTorrent, um repositório de dados de projetos de software de código aberto que usam o Travis CI como sistema de integração contínua. O estudo conclui que a Integração Contínua pode ter um impacto positivo na prática de refatoração, pois fornece *feedback* rápido e frequente aos desenvolvedores sobre a qualidade do código. No entanto, o estudo também destaca que a Integração Contínua pode aumentar a carga cognitiva dos desenvolvedores, pois eles precisam lidar com *feedback* constante e rápido. O estudo sugere que a adoção da Integração Contínua deve ser acompanhada de práticas de gerenciamento de *feedback* para minimizar a carga cognitiva dos desenvolvedores.

O trabalho de Soares *et al.* (2023) é um estudo que investiga a relação entre a integração contínua e a qualidade do software. O estudo utiliza a técnica de Grafos Acíclicos Direcionados (DAGs) para analisar a relação causal entre as variáveis. O estudo também identifica a falta de conhecimento em programação e estratégia e a falha em identificar as características

específicas dos problemas como possíveis causas humanas para problemas de software.

O estudo de Santos *et al.* (2022) investiga o impacto da IC na produtividade e qualidade de projetos de código aberto. A pesquisa revela que projetos com os melhores valores para as subpráticas de IC enfrentam menos problemas relacionados à IC em comparação com projetos que exibem os piores valores para essas subpráticas. Conclui-se que os projetos devem se esforçar para manter várias subpráticas de IC, pois estas podem impactar na produtividade e qualidade dos projetos. Além disso, uma análise quantitativa demonstrou que a melhoria da maturidade da automação de testes pode resultar em melhorias na qualidade do produto.

Em seu trabalho Freitas *et al.* (2023) investigam o impacto das subpráticas de IC na qualidade do código em projetos de código aberto. O estudo revela que a qualidade da inspeção é melhorada em projetos com maior cobertura de testes e duração de compilação mais curta. Além disso, a pesquisa identifica subpráticas específicas de IC, como manter durações de compilação mais curtas, que apresentam uma forte correlação com a melhoria da qualidade da inspeção. Os resultados indicam que simplesmente adotar um serviço de IC não garante melhores resultados de qualidade, sendo essencial que os desenvolvedores sigam consistentemente as subpráticas recomendadas de IC para colher os benefícios da integração contínua em seus projetos. O estudo destaca a importância de aderir a essas subpráticas para alcançar melhores resultados de qualidade.

Em seu trabalho Saraiva *et al.* (2023) investigam a relação entre a IC e a cobertura de código em projetos de software. Os resultados indicam que a adoção da IC está associada a um aumento na cobertura de código, o que sugere que a prática da IC pode contribuir para a melhoria da qualidade do software por meio da automação de compilação e testes. Além disso, o estudo destaca a importância de práticas fundamentais da IC, como *commits* frequentes, *builds* frequentes e testes automatizados, para obter benefícios significativos. No entanto, também aponta que muitos projetos adotam a IC sem dedicar atenção suficiente a essas práticas, o que pode resultar em um fenômeno denominado "Teatro de Integração Contínua". Portanto, o trabalho fornece *insights* valiosos sobre a relação entre IC e cobertura de código, destacando a importância de adotar práticas de IC de forma abrangente para obter os benefícios esperados.

Nesta seção foram apresentados alguns trabalhos existentes que abordam a questão da IC e qualidade de *software*. Dentre os resultados verificou-se que o processo de IC agrega valor à qualidade final de um sistema. A proposta do presente trabalho se assemelha com os estudos apresentados pois também trata da temática de IC e qualidade de *software*, entretanto

o objetivo aqui é verificar se, em contraste com as boas práticas de IC, as más práticas de IC podem afetar negativamente a qualidade de *software*.

3.4 Integração contínua e revisão de código

Na literatura, ainda há poucos estudos investigativos sobre a relação entre as práticas de RC e IC (Rahman; Roy, 2017; Chen *et al.*, 2019; Zampetti *et al.*, 2019; Wessel *et al.*, 2020; Cassee *et al.*, 2020).

Rahman e Roy (2017) realizaram um estudo exploratório quantitativo analisando os *logs* registrados de milhares de *builds* automatizados realizados em projetos *open-source* no GitHub. Os autores analisaram dois aspectos da IC - *status* e frequência do *build* automatizado e dois aspectos da RC - participação na revisão e a qualidade da revisão. Foi realizada uma correlação destes aspectos, e as constatações foram que os *builds* executados com sucesso são mais propensos a encorajar a participação dos revisores através de novas revisões de código em *pull request*.

Paixao *et al.* (2018) abordam a problemática relacionada ao fato de os repositórios de código não permitirem de forma simples que as revisões de código sejam relacionadas com as respectivas versões do sistema. Os autores desenvolveram a ferramenta CROP ("***C**ode **R**eview **O**pen **P**latform*"), uma plataforma que consegue ligar as revisões de código ao determinado ponto do sistema no momento da revisão. De forma similar a Paixao *et al.* (2018), neste trabalho as revisões de código também são analisadas através de uma ferramenta de *software*, mais especificamente um *script* desenvolvido para extrair métricas a partir de dados de RC.

Zampetti *et al.* (2019) conduziram um estudo empírico qualitativo para descobrir como as equipes de desenvolvimento utilizam o resultado dos *builds* de IC durante as atividades de RC. Os autores analisaram um total de 64.865 discussões relacionadas às solicitações de integração no ambiente de 69 projetos de código-fonte aberto. A pesquisa foi complementada através de uma consulta a 13 desenvolvedores. O estudo indicou que os pedidos de integração onde os *builds* de IC são aprovados possuem maior probabilidade de serem integrados do que aqueles onde o *build* falha. Além disso, os participantes consultados informaram que mesmo solicitações de integração com falha de *build* podem ser integradas. Isto ocorre devido ao fato de algumas falhas ocorrerem por "pequenos problemas" de análise estática, por exemplo, que os desenvolvedores preferem ignorar ou resolver posteriormente.

Chen *et al.* (2019) conduziram um estudo que combina técnicas de *crowdsourcing* e

data mining para replicar estudos qualitativos sobre *pull requests*. Os autores analisaram um conjunto de 190 *pull requests* de 142 projetos no GitHub. Para este fim, os autores construíram uma ferramenta para prever se o código seria integrado ou não com base em métricas quantitativas e opiniões qualitativas dos desenvolvedores sobre o código. No presente trabalho, foram selecionadas algumas métricas do conjunto apresentado por Chen *et al.* (2019) para implementação em um *script* de modo a permitir a coleta automática a partir de projetos industriais no GitLab. Dessa forma, aspectos da participação da equipe na RC e características do processo de CI nas solicitações de integração poderão ser analisados de forma objetiva.

Cassee *et al.* (2020) conduziram um estudo exploratório quantitativo do impacto da IC na RC em projetos de código aberto. Os autores partem da premissa de que o emprego da IC permite que os revisores se concentrem em aspectos mais complexos da qualidade de *software* que não poderiam ser capturados somente com a IC. Para validar esta suposição foi realizado um estudo exploratório a partir de 685 revisões em projetos *open-source* no GitHub que utilizaram o Travis CI como servidor de IC. As conclusões do estudo foram: (1) com a introdução de IC, os *pull requests* passaram a ser menos discutidos; (2) a IC economiza pelo menos um comentário de revisão por *pull request*; (3) desenvolvedores que trabalham com IC entregam em geral a mesma quantidade de trabalho mas discutem menos as modificações.

Wessel *et al.* (2020) conduziram um estudo para investigar os efeitos da adição de *bots* de IC para realizar comentários de revisão dentro de *pull requests* em projetos de código aberto usando o ambiente do GitHub. Como resultado os autores descobriram que a adoção de *bots* afeta algumas métricas, como por exemplo: (1) aumento no número de *pull requests* integrados mensalmente; (2) diminuição no número de *pull requests* não integrados mensalmente; e, (3) diminuição no volume de comunicação entre os membros da equipe de desenvolvimento. No presente trabalho de dissertação, serão analisadas as solicitações de integração de projetos de código fechado no GitLab. Entretanto, o foco da investigação estará relacionado à intensidade da participação de membros reais da equipe de desenvolvimento (em vez de *bots*) nas tarefas de RC e IC.

Nesta seção foram apresentados estudos que relacionam as práticas de IC com a aplicação da RC sobretudo em projetos *open-source*. Diversos resultados sugerem que a IC, embora introduza uma complexidade ao processo de RC, consegue gerar bons resultados pois atua na diminuição da carga de trabalho imposta ao revisores ao mesmo tempo que atua na garantia da qualidade de *software*. Sendo assim, a proposta do presente trabalho é verificar se

as más práticas de IC podem impactar negativamente na realização das atividades de RC no contexto de projetos industriais.

3.5 Comparação dos trabalhos relacionados

A Tabela 5 apresenta os trabalhos relacionados a esta proposta de dissertação, bem como as semelhanças e diferenças de cada um. Para melhor entendimento foram especificados alguns atributos presentes em cada estudo: (1) Tipo de Projeto, diz se o trabalho em questão aborda projetos *open-source*, industriais ou ambos; (2) Más Práticas de IC, diz respeito a se o estudo analisou a problemática das más práticas; (3) Percepção dos Desenvolvedores, indica se o estudo considerou as opiniões das equipes de desenvolvimento para obter seus resultados; e (4) Característica Avaliada, apresenta a principal característica analisada durante o estudo. Através da análise dos trabalhos relacionados é possível perceber que poucos deles abordam a questão das más práticas de IC em projetos de *software*, sobretudo no contexto dos projetos industriais. Sendo que, a grande maioria só compreende as boas práticas de IC. O mesmo ocorre nas revisões sistemáticas da literatura apresentadas, que também priorizam a análise das práticas de IC e seus efeitos no ciclo de vida do desenvolvimento de *software*. Vale ressaltar, que alguns dos trabalhos relacionados tentam estabelecer uma relação entre a IC e a qualidade do produto final, bem como com o processo de RC. Entretanto, não foram encontrados estudos específicos com relação às más práticas de IC. Assim, no presente trabalho será realizada uma investigação acerca de como os efeitos das más práticas de IC que são percebidas pelas equipes de desenvolvimento e como eles afetam a qualidade do *software* e o processo de RC no contexto de projetos industriais.

3.6 Conclusão

Neste capítulo foram apresentados os estudos relacionados a este trabalho. Foram identificadas lacunas em aberto e deficiências dos estudos no que diz respeito à investigação do impacto das más práticas de IC em projetos industriais.

As revisões da literatura apresentadas na Seção 3.1 foram importantes para identificar o estado da arte no tocante às práticas de IC, o que permitiu verificar as lacunas de pesquisa existentes, sobretudo com relação às más práticas de IC. Nas Seções seguintes, foram apresentados trabalhos que abordam os conceitos referentes às más práticas de IC bem como o relacionamento entre a IC, qualidade de *software* e o processo de RC. Constatou-se que existem

Tabela 5 – Comparação dos trabalhos relacionados

TRABALHOS	TIPO DE PROJETO	MÁS PRÁTICAS DE IC	PERCEPÇÃO DOS DESENVOLVEDORES	CARACTERÍSTICA AVALIADA
"Proposta do autor"	Industrial	Sim	Sim	Efeitos das más práticas de IC em projetos de software
Felidré *et al.* (2019)	*Open-source*	Sim	Não	Más práticas de IC
Vassallo *et al.* (2019)	*Open-source*	Sim	Sim	Más práticas de IC
Gallaba e McIntosh (2020)	*Open-source*	Sim	Não	Más práticas de IC
Zampetti *et al.* (2020)	*Ambos*	Sim	Sim	Más práticas de IC
Elazhary *et al.* (2021)	Industrial	Sim	Sim	Más práticas de IC
Hamdan e Alramouni (2015)	Industrial	Não	Não	Atributos de qualidade
Alizadeh *et al.* (2019)	Ambos	Não	Sim	Qualidade de código
Freitas (2020)	*Open-source*	Não	Não	Qualidade de código
Saidani *et al.* (2021)	*Open-source*	Não	Sim	Qualidade de código
Soares *et al.* (2023)	*Open-source*	Não	Não	Qualidade de código
Santos *et al.* (2022)	*Open-source*	Não	Não	Qualidade de código
Freitas *et al.* (2023)	*Open-source*	Não	Não	Qualidade de código
Saraiva *et al.* (2023)	*Open-source*	Não	Não	Cobertura de código
Lai *et al.* (2021)	Industrial	Não	Sim	Qualidade de manutenção
Mohammad (2016)	Ambos	Não	Não	Qualidade de manutenção
Rahman e Roy (2017)	*Open-source*	Não	Não	Revisão de código
Paixao *et al.* (2018)	*Open-source*	Não	Não	Revisão de código
Chen *et al.* (2019)	*Open-source*	Não	Sim	Revisão de código
Zampetti *et al.* (2019)	*Open-source*	Não	Sim	Revisão de código
Wessel *et al.* (2020)	*Open-source*	Não	Não	Revisão de código
Cassee *et al.* (2020)	*Open-source*	Não	Não	Revisão de código

Fonte: Elaborado pelo autor.

lacunas de pesquisa quanto ao relacionamento das más práticas de IC: (1) percepção das equipes de desenvolvimento com relação às más práticas de IC; (2) correlação entre as más práticas de IC e a qualidade de *software*; e (3) efeitos das más práticas de IC no processo de RC. Estas três áreas constituem a base para esta dissertação.

4 ANALISANDO MÁS PRÁTICAS DE INTEGRAÇÃO CONTÍNUA EM PROJETOS DE CÓDIGO FECHADO: UM ESTUDO INICIAL

Este capítulo apresenta um estudo investigando o impacto das más práticas de IC em dois projetos industriais. O estudo foi publicado na trilha de ideias e resultados emergentes do 34º Simpósio Brasileiro de Engenharia de Software (SBES) 2020 (Silva; Bezerra, 2020).

4.1 Introdução

A IC é um processo que surgiu com o objetivo de promover a qualidade do software mediante a adoção de boas práticas relacionadas ao ciclo de vida de desenvolvimento. Dentre estas práticas, podemos destacar a utilização de um repositório central para armazenar as versões do código, integração constante das novas modificações à *branch* principal, uso de testes automatizados e análise estática do código (Beller *et al.*, 2017; Felidré *et al.*, 2019).

Ao longo dos anos, alguns estudos revelaram que a IC consegue trazer diversos benefícios tanto para projetos *open-source* quanto industriais que adotam esta metodologia (Goodman; Elbaz, 2008; Miller, 2008; Downs *et al.*, 2010; Vasilescu *et al.*, 2015; Beller *et al.*, 2017). Talvez por conta disto, muitas organizações têm aderido ao uso das boas práticas de IC em seus projetos (Rebouças *et al.*, 2017; Pinto *et al.*, 2018).

Alguns estudos relatam a ocorrência de problemas no ambiente de desenvolvimento que podem prejudicar a eficiência do processo de IC, de forma que os benefícios não ocorram adequadamente (Ståhl; Bosch, 2014; Shahin *et al.*, 2017; Ståhl *et al.*, 2017; Vassallo *et al.*, 2018; Vassallo *et al.*, 2019). No presente trabalho, estes problemas são caracterizados como "Más práticas de IC". Estas más práticas foram observadas por Zampetti *et al.* (2020) que constataram que elas podem prejudicar o projeto como um todo. Felidré *et al.* (2019) enfatizam que a adoção de IC não necessariamente resulta em benefícios, segundo os autores as más práticas criam um pseudoprocesso de IC que chamaram de "teatro de integração contínua".

Dessa forma, este estudo realiza uma investigação inicial dos efeitos causados pelas más práticas em IC no contexto de projetos industriais. Para realização do estudo, foram utilizadas as más práticas de IC catalogadas no trabalho de (Zampetti *et al.*, 2020), que fornece um conjunto de 79 *bad smells* de IC divididos em 7 categorias. As más práticas de IC foram investigadas em 2 projetos industriais, analisando: (1) a frequência que as más práticas de IC ocorrem nos projetos, (2) quanto tempo as falhas de *build* permanecem não resolvidas; e (3) os efeitos das más práticas de IC na saúde do software na perspectiva dos desenvolvedores e

gerentes de projetos.

4.2 Projeto do estudo

4.2.1 Objetivos e questões de pesquisa

O principal objetivo deste estudo é analisar más práticas de IC no contexto de projetos industriais, utilizando o catálogo de más práticas de IC do estudo de Zampetti *et al.* (2020). Para atingir este objetivo, foram elaboradas três questões de pesquisa para guiar o presente estudo.

- **QP**$_1$ *Quais são as más práticas de IC mais frequentes em projetos da perspectiva de desenvolvedores e gerentes de projeto?* A QP$_1$ busca identificar quais são as más práticas de IC percebidas com mais frequência por desenvolvedores e gerentes de projeto. Ao responder a QP$_1$ será possível saber as áreas de concentração destas más práticas e, também, abrirá caminho para futuras pesquisas no tópico.
- **QP**$_2$ *Por quanto tempo falhas de build permanecem sem solução nos projetos?* QP$_2$ busca investigar qual abordagem é utilizada para lidar com falhas de *build*. Desta maneira, será possível entender melhor os procedimentos de IC na prática. O histórico de execução de *build* dos sistemas investigados foi analisado. A resposta à esta questão de pesquisa pode também ampliar a visão sobre pesquisas direcionadas aos dados disponíveis em sistemas de controle de versão.
- **QP**$_3$ *Quais são os efeitos de más práticas de IC na saúde do software da perspectiva de desenvolvedores e gerentes de projeto?* A QP$_3$ destina-se a analisar e especificar os efeitos negativos na saúde do software causados pela ocorrência de más práticas de IC. Esta questão de pesquisa pode apontar o comportamento do time (tanto de desenvolvedores quanto de gerentes) no que se refere às más práticas de IC nos dois projetos em foco no presente estudo.

4.2.2 Passos do estudo

Esta Seção descreve as etapas do estudo, a fim de apoiar a investigação. Os passos descritos a seguir foram elaborados de modo a responder as questões de pesquisa.

Passo 1: Selecionar sistemas de software para análise. Foram selecionados 2 sistemas de software, escritos na linguagem Java, e de código fechado que estão sendo desenvolvidos pelos parceiros industriais. Os critérios utilizados para seleção dos projetos foram: (1) sistemas

com maior quantidade de linhas de código; (2) sistemas que não estavam em suas versões iniciais; (3) sistemas escritos na linguagem Java; e (4) sistemas que tenham sido configurados para análise estática e testes automatizados, caso existam. A Tabela 12 apresenta os dados gerais dos sistemas por ID, domínio, número de linhas de código (LoC) e se o sistema possui ou não testes. Estes sistemas foram escolhidos por possuírem a configuração de IC com fases de *build*/testes, análise estática e *deploy*. Com relação aos testes, foi considerada a presença de algum tipo de teste funcional independente do tipo (caixa branca, caixa preta, regressão, unitário, integração, etc.).

Tabela 6 – Dados gerais dos sistemas de software alvos

SISTEMA	DOMÍNIO	TAMANHO	POSSUI TESTES
S1	Prontuário odontológico eletrônico	7830	SIM
S2	Proficiência leitora em idioma estrangeiro	15269	NÃO

Fonte: Elaborado pelo autor.

Passo 2: Elaborar e aplicar o questionário. Foi elaborado um questionário [1] composto por perguntas fechadas (análise quantitativa) e perguntas abertas (análise qualitativa). Deste modo, as perguntas fechadas serviram para responder a QP_1 e as perguntas abertas para responder a QP_3.

Foram importadas 79 más práticas de IC identificadas no trabalho de Zampetti *et al.* (2020). A seguir, foram criadas 7 perguntas fechadas e de múltipla escolha. Para cada má prática, o respondente selecionou apenas uma das opções listadas abaixo, no que diz respeito à existência (ou não) e ao impacto negativo das más práticas de IC: (1) Não sei informar; (2) Não existe no projeto; (3) Existe, mas não tem impacto; (4) Existe e impacta levemente; e, (5) Existe e impacta fortemente.

As 7 questões do questionário representam as categorias de más práticas listadas na Tabela 7. As más práticas foram separadas nas questões de acordo com sua categoria. Além disso, foram desenvolvidas duas questões abertas, a primeira para captar a experiência do respondente em relação aos benefícios das boas práticas de IC e a segunda, para que o respondente pudesse listar as perdas causadas pelas más práticas de IC. Foi enviado um *e-mail* com o questionário *online* aos participantes de ambos os projetos, tanto desenvolvedores quanto gerentes de projeto.

A Tabela 8 mostra o perfil dos respondentes do questionário. A tabela foi dividida da seguinte forma: **ID** - identificação do participante; **Sistema** - sistema em que o participante está alocado; **Papel** - papel do participante no projeto; **Tempo Exp.** - tempo de experiência em anos em desenvolvimento de sistemas; **Nível** - nível de escolaridade do participante; e, **IC** - nível

[1] Ver o Apêndice A.

Tabela 7 – Categorias de más práticas de IC (Zampetti *et al.*, 2020)

ID	CATEGORIA	DESCRIÇÃO
REP	Repositório	Uso indevido de repositório e sistema de controle de versão
INF	Escolhas de infraestrutura	Más escolhas de componentes de *hardware* e *software* para o *pipeline* de IC
OPB	Organização do processo de *build*	Má configuração do *pipeline* de IC
MB	Manutenibilidade do *build*	Capacidade de manutenção do *pipeline* de IC
GQ	Garantia da qualidade	Etapas de teste e análise estática
PE	Processo de entrega	Gerenciamento dos artefatos de liberação do projeto
CUL	Cultura	Fator humano e cultura

Fonte: Elaborado pelo autor.

de conhecimento do participante em IC. Foi obtido um total de 10 respostas, 2 participantes com a função de gerente de projeto e 8 desenvolvedores. Em relação à alocação, 4 dos respondentes relataram trabalhar no sistema S1 e 6 no sistema S2. Também é possível observar na Tabela 8 que apenas 2 pessoas possuem o título de mestre, o restante dos participantes possui apenas o nível de graduação. Por fim, quanto ao nível de experiência em IC, apenas 3 participantes afirmaram possuir conhecimento intermediário e o restante um nível de conhecimento básico.

Tabela 8 – Caracterização dos respondentes

ID	SISTEM	PAPEL	TEMPO DE EXP.	NÍVEL	IC
P01	S1	Gerente de projetos	11	Mestre	Básico
P02	S2	Gerente de projetos	10	Mestre	Intermediário
P03	S1	Desenvolvedor	5	Graduação	Intermediário
P04	S2	Desenvolvedor	4	Graduação	Básico
P05	S2	Desenvolvedor	3	Graduação	Básico
P06	S2	Desenvolvedor	2	Graduação	Básico
P07	S1	Desenvolvedor	1	Graduação	Intermediário
P08	S2	Desenvolvedor	1,5	Graduação	Básico
P09	S1	Desenvolvedor	1	Graduação	Básico
P10	S2	Desenvolvedor	1	Graduação	Básico

Fonte: Elaborado pelo autor.

Passo 3: Analisar as respostas do questionário As respostas foram analisadas considerando os tipos de perguntas.

- **perguntas fechadas** - para cada um dos sistemas foram identificadas as más práticas observadas pelos respondentes e foi realizada a ordenação das mesmas de acordo com o número de vezes que ela apareceu nas respostas. Para cada má prática que ocorreu pelo menos uma vez no decorrer do questionário foi catalogada também a porcentagem de quantas vezes a má prática foi identificada como "existindo mas não tendo impacto", "existindo e impactando levemente" e "existindo e impactando fortemente".
- **perguntas abertas** - primeiramente foi realizada a leitura das respostas das duas perguntas abertas, e em seguida foi realizado um tratamento de forma diferente para cada pergunta: **benefícios das boas práticas** - extração do "efeito oposto" ao benefício, ou seja, a partir

dos ganhos obtidos pelas boas práticas foram deduzidas as perdas no caso em que elas não fossem aplicadas adequadamente; **prejuízos das más práticas** - extração de forma literal do efeito negativo relatado pelo respondente. Desta forma foi possível catalogar de forma mais abrangente os prejuízos causados pelas más práticas de IC.

Passo 4: Análise do histórico de pipeline para sistemas Finalmente, para responder à QP_2, foi realizada uma análise do histórico de execução do *pipeline* para ambos os sistemas. O histórico em formato JSON foi obtido acessando a API do sistema de IC utilizado pelos sistemas. Para facilitar a análise, uma ferramenta[2] foi implementada para realizar a importação de dados. A ferramenta procedeu com a análise da seguinte forma: (1) primeiro, os *pipelines* foram divididos de acordo com o *branch* para o qual foram executados; (2) então os *pipelines* que falharam foram identificados e o tempo que levou para um novo *pipeline* ser executado com sucesso foi calculado (i.e., o tempo em que uma construção defeituosa foi corrigida foi calculado); (3) em seguida, foi contabilizado o cálculo dos tempos médios de correção dos *pipelines*; e, (4) finalmente, os resultados foram disponibilizados em um relatório que continha os intervalos de correção e sua duração, a duração do intervalo mais longo, a duração do intervalo mais curto e a duração média de todos os intervalos[3].

4.3 Resultados

4.3.1 Más práticas de IC mais frequentes (QP_1)

A QP_1 foi respondida identificando as más práticas de IC que mais ocorreram nos dois projetos analisados. A Tabela 9 lista as más práticas de IC que foram mais observadas pelos respondentes no formulário, os dados de ambos os sistemas foram considerados em conjunto. A primeira coluna apresenta a categoria de cada má prática, a segunda coluna descreve as más práticas, a terceira coluna lista o número de ocorrências para cada uma, e as três colunas restantes expressam, respectivamente, a porcentagem de respondentes que indicaram que a má prática "existe mas não impacta" (ENI), "existe e impacta levemente" (EIL) e "existe e impacta fortemente" (EIF).

Na Tabela 9, pode ser constatado que as más práticas de IC mais citadas pelos respondentes estão relacionadas ao gerenciamento do repositório, ao sistema de controle de

[2] A ferramenta está disponível no link: https://github.com/ruben-silva-dev/dissertacao-pcomp-ic/blob/main/estudo-1/pipeline-api-tool.zip

[3] Os dados coletados estão disponíveis no seguinte link: https://github.com/ruben-silva-dev/dissertacao-pcomp-ic?tab=readme-ov-file#artefatos-do-primeiro-estudo

Tabela 9 – Más práticas de IC mais frequentes

CAT	MÁ PRÁTICA	OC	ENI	EIL	EIF
REP	*Feature branches* são usadas em vez de *feature toggles*	7	∼28,57%	∼42,85%	∼14,28%
REP	*Branches* divergentes	7		∼42,85%	∼57,14%
CUL	Os desenvolvedores não têm controle total do ambiente	7	∼42,85%	∼42,85%	∼14,28%
BPO	Notificações de falhas são enviadas apenas para equipes/desenvolvedores alocados na tarefa	6	∼33,33%	∼33,33%	∼33,33%
QA	Testes ausentes em *feature branches*	6		∼33,33%	∼66,66%
BPO	Algumas tarefas de *pipelines* são iniciadas manualmente	5	80,00%	20,00%	
QA	Falta de teste em um ambiente de produção	5		40,00%	60,00%
REP	Os casos de teste não são organizados em pastas com base em seus propósitos	4	25,00%	50,00%	25,00%
INF	O hardware do servidor IC é usado para finalidades diferentes, além da execução da estrutura de IC	4	25,00%	75,00%	
INF	Ferramentas externas são usadas com suas configurações padrão	4	50,00%	25,00%	25,00%
BPO	Um *build* falha devido a alguma instabilidade na execução que não deveria ocorrer	4	25,00%	75,00%	
BM	Scripts de *build* longos	4	75,00%	25,00%	
QA	*Quality gates* são definidos sem os desenvolvedores, considerando apenas o que é ditado pelo cliente	4	25,00%	50,00%	25,00%
CUL	Desenvolvedores e operadores são mantidos como funções separadas	4		75,00%	25,00%

Fonte: Elaborado pelo autor.

versão e a cultura do ambiente de trabalho. As más práticas de CI mais frequentes foram: "*Feature branches* são usadas em vez de *feature toggles*", "*Branches* divergentes" e "Os desenvolvedores não têm controle completo do ambiente".

Descoberta 1 *As más práticas de IC mais frequentes estão relacionadas ao gerenciamento de repositório e à cultura do ambiente.*

Outro ponto é que todas as más práticas listadas foram sinalizadas como tendo pelo menos um impacto ligeiramente negativo. No entanto, 3 das más práticas listadas foram avaliadas pela maioria dos entrevistados como tendo um forte impacto negativo: "*Branches* divergentes", "Testes ausentes em *feature branches*" e "Falta de teste em um ambiente de produção". Essas más práticas estão relacionadas ao sistema de gerenciamento de repositório e controle de versão; e, com garantia de qualidade.

Descoberta 2 *As más práticas de IC com maior impacto negativo estão relacionadas ao gerenciamento de repositório e garantia de qualidade.*

4.3.2 Tempo necessário para corrigir falhas de build (QP_2)

A QP_2 foi respondida analisando a duração dos intervalos entre uma *build* com falha e uma *build* executada com sucesso, o que é caracterizado aqui como "intervalo de correção de *build* ". A Tabela 10 mostra o resultado da análise dos tempos de correção para *builds* com

falha para os sistemas S1 e S2, que foram analisados neste trabalho. Um total de 80 intervalos de correção de *build* foram encontrados para o sistema S1 e 96 para o sistema S2. É possível observar na Tabela que, para ambos os sistemas, o percentual de intervalos de mais de 3 horas de duração ultrapassa 50%.

Tabela 10 – Tempo de resolução das falhas de *build*

PROPRIEDADE	S1	S2
Pipelines analisados	1036	1256
Quantidade de intervalos	80	96
% tempo >3h	53,75%	61,46%
% tempo <= 3h	46,25%	38,54%
Maior duração	206d 21h 40m 27s	84d 21h 56m 22s
Duração mais curta	02m 09s	01m 01s
Duração média	10d 23h 25m 16s	5d 15h 50m 49s
Desvio padrão	∼32d 14h 32m 06s	∼11d 16h 57m 33s

Fonte: Elaborado pelo autor.

Outro fato curioso é que existe uma diferença considerável entre os limites superior e inferior da duração dos intervalos de correção de *build*. Como é possível ver na Tabela 10, o menor e o maior tempo de correção para o sistema **S1** foi "02m 09s" e "206d 21h 40m 27s", respectivamente. Para o sistema **S2**, o tempo de correção mais curto e mais longo foi "01m 01s" e "84d 21h 56m 22s". Considerando todos os registros, o tempo médio de correção de construção foi "10d 23h 25m 16s" para o sistema **S1** e "5d 15h 50m 49s" para o sistema **S2**. Quanto ao desvio padrão, o cálculo revela aproximadamente "32d 14h 32m 06s" para o sistema **S1** e "11d 16h 57m 33s" para o sistema **S2**. Esses dados permitem afirmar que embora a duração média dos intervalos de correção aponte para o mesmo dia, a maioria das correções é adiada, podendo mesmo se estender por vários meses.

Descoberta 3 *A maioria das falhas de compilação não é resolvida imediatamente, algumas podem levar meses para serem corrigidas.*

4.3.3 Efeitos das más práticas de IC na saúde do software (QP_3)

A QP_3 foi respondida através de uma análise de caráter qualitativo dos efeitos negativos provocados pelas más práticas de IC. A Tabela 11 lista todas as observações de problemas decorrentes das más práticas de IC que foram relacionadas pelos respondentes do

questionário. A primeira coluna descreve a dificuldade relatada, a segunda coluna enumera quantas vezes a dificuldade foi reportada, e a terceira coluna mostra a categoria da dificuldade. As categorias das dificuldades aqui mencionadas diferem daquelas encontradas na Tabela 7, pois estão relacionadas às dificuldades relatadas pelos respondentes e não às más práticas de IC.

Tabela 11 – Dificuldades causadas pelas más práticas de IC

DIFICULDADE	#	CATEGORIA
Maior tempo e custo para desenvolver e implantar uma funcionalidade	15	Desenvolvimento
A falta de medição contínua de qualidade pode encorajar os desenvolvedores a aplicar técnicas de programação ruins	11	Desenvolvimento
A equipe pode se sentir insegura por não trabalhar totalmente e por ter que se preocupar com recursos defeituosos, perdendo assim o foco no desenvolvimento	10	Desenvolvimento
Maior probabilidade de propagação de erros, recursos mal implementados e problemas de segurança	9	Garantia da qualidade
Diminuição da produtividade da equipe devido à execução de muitas atividades de forma manual	7	Desenvolvimento
A falta de modularização e padrões organizacionais pode gerar um código que é difícil de entender e com baixa manutenção	6	Garantia da qualidade
Alto custo de integração do trabalho de diferentes desenvolvedores	6	Gerenciamento de mudanças
Feedback não confiável durante o processo de construção, levando a mudanças desnecessárias no código-fonte do software	5	Desenvolvimento
Maior nível de esforço para rastrear a qualidade das mudanças no software	4	Garantia da qualidade
Insegurança para implantação na produção devido a falhas de automação	4	Garantia da qualidade
Usar o IC sem testes automatizados pode introduzir erros/falhas nas funcionalidades existentes, prejudicando a garantia de qualidade	4	Garantia da qualidade
Falhas de comunicação entre a equipe que podem levar a confusão sobre as más práticas de IC	3	Comunicação
Dificuldade de gestão e maior complexidade para tomada de decisão técnica em relação ao projeto	2	Gerenciamento de projetos
Retorno de *feedback* lento e acumulação de código local pendente, o que pode causar um grande número de conflitos com o código principal, aumentando o tempo, a complexidade para correção, prejudicando a saúde do software	2	Gerenciamento de mudanças
A falha no mapeamento e/ou detecção de falhas deixa a equipe refém do *feedback* do usuário, o que pode comprometer a reputação do software e da equipe de desenvolvimento	1	Garantia da qualidade

Fonte: Elaborado pelo autor.

Através da análise das dificuldades relatadas é possível verificar que os participantes possuem uma compreensão abrangente acerca dos prejuízos decorrentes das más práticas de IC. As respostas obtidas estão mais fortemente relacionadas com os aspectos de desenvolvimento e com a garantia da qualidade do software. Alguns fatores foram também observados, como: (1) a percepção acerca das perdas sobre a produtividade, (2) a dificuldade de gerenciamento do projeto e (3) os obstáculos para implantação do sistema.

Descoberta 4 *Existe a percepção da equipe em relação às perdas causadas pelas más práticas de IC.*

4.4 Discussão

Através dos resultados obtidos neste estudo, é possível entender um pouco melhor a natureza dos impactos das más práticas de IC em sistemas industriais. Existe certa euforia com relação ao uso da IC em projetos de software, entretanto, como foi possível observar na nossa análise, os tão pregados benefícios da IC podem ser invalidados caso não seja dada a devida atenção ao uso correto deste processo.

Em uma análise mais apurada da frequência de ocorrência das más práticas de IC, foi possível constatar que as mais proeminentes dizem respeito a um aspecto relativamente simples no contexto da IC: o gerenciamento do repositório. Isto revela que os processos de IC podem estar sendo usados ainda de forma imatura. O comentário de um dos respondentes em uma de suas respostas confirma isto: *"Algumas questões de versionamento, estrutura de atividades contínuas e integradas deveriam ser mais debatidas com os desenvolvedores que muitas vezes não tem noção do que deve fazer para desenvolver utilizando boas práticas"*.

Comparando com o trabalho Zampetti *et al.* (2020), é possível ver que o método de análise das más práticas de IC utilizado no presente trabalho é semelhante. Entretanto, neste trabalho foi levado em conta (1) o impacto negativo das más práticas em vez do seu nível de relevância, e (2) devido ao tamanho reduzido da amostra utilizada neste trabalho não é possível aplicar a generalização nos resultados.

A duração dos intervalos entre um *build* defeituoso e um *build* que executou com sucesso é um ponto que chama atenção. O fato é que a grande maioria das correções demoram demais para serem implementadas, e certamente podem ocorrer casos em que o conserto, não foi efetuado[4]. Isto permite fazer algumas suposições: (1) algumas *branches* são ignoradas; (2) o conteúdo das *branches* principais não é atualizado constantemente, com isso as correções demoram a aparecer nas *pipelines* de IC; (3) o fato de muitas correções demorarem a acontecer pode revelar um gerenciamento deficiente das atividades alocadas para os desenvolvedores (e.g., com tarefas muito extensas); e, (4) as correções podem ter sido na verdade pseudocorreções (e.g., adequação dos *thresholds* para o estado atual do sistema apenas para o *build* ser executado com sucesso).

Com relação aos efeitos nocivos das más práticas de IC na saúde do software, fica claro que as dificuldades listadas na Tabela 11 referem-se a aspectos de desenvolvimento de software e garantia de qualidade. Como esperado, foi relatado que as más práticas de IC

[4] Na análise do histórico de *builds* não foram considerados os casos em que as falhas de *build* não foram resolvidas.

interferem principalmente no tempo e no custo necessários para entregar o produto final ao cliente. Um fato importante que se pôde observar é que, quando não configuradas corretamente, as ferramentas de IC podem gerar perdas devido ao levantamento de pseudo erros que causam alterações desnecessárias no código-fonte.

4.5 Ameaças à validade

Esta seção discute ameaças à validade do estudo de acordo com a classificação de Wohlin *et al.* (2012).

Validade interna. Uma das ameaças à validade interna é a falta de experiência dos participantes do projeto com IC e más práticas de IC. Porém, os participantes tiveram muito contato com o ambiente de IC dos projetos aos quais foram alocados.

Validade de construção. A identificação dos intervalos para correção de *builds* defeituosos foi realizada automaticamente através de uma ferramenta desenvolvida durante este estudo, o que ajuda a diminuir a chance de erros. No entanto, a ferramenta pode não ter considerado alguma situação e algum intervalo pode ter sido deixado de fora de nossa análise.

Validade externa. Uma ameaça à validade externa é o uso de apenas dois sistemas na linguagem Java para realizar o estudo. Em um estudo com um número mais significativo de sistemas, os resultados da análise dos intervalos de correção de *build* defeituosos podem variar. No entanto, foi observado que ambos os sistemas têm um número considerável de *logs* de construção, o que foi suficiente para a investigação. Além disso, o uso de sistemas em outros idiomas pode ter um resultado diferente. Outra ameaça à validade externa é que nem todos os desenvolvedores dos dois sistemas responderam ao questionário. No entanto, foram obtidas respostas da maioria dos desenvolvedores. Finalmente, outra ameaça à validade é o pequeno tamanho das equipes. No entanto, isso ocorre porque estão em um ambiente de cultura ágil que valoriza equipes pequenas e multidisciplinares.

Validade da conclusão. As más práticas de IC usadas Zampetti *et al.* (2020) continham alguns termos complicados na área de IC, o que poderia causar uma interpretação incorreta das práticas inadequadas e prejudicar os resultados. Para mitigar essa ameaça, foi realizada uma operação de adaptação dos termos inicialmente utilizados pelos autores e também foram adicionadas observações sobre alguns deles, preservando o significado original para melhorar o entendimento.

4.6 Conclusão

Este capítulo apresentou um estudo inicial para investigar as más práticas de IC em dois projetos industriais. O objetivo principal deste trabalho foi diversificado em 3 questões de pesquisa para descobrir: (i) quais são as práticas inadequadas de IC mais comuns em projetos, (ii) por quanto tempo as falhas de construção permanecem sem solução e (iii) quais são os efeitos das práticas inadequadas de IC na integridade do software. Os resultados foram obtidos por meio da aplicação de um questionário com a equipe do projeto e da análise do histórico.

As principais conclusões foram que as más práticas de IC mais frequentes estão ligadas a fatores de gerenciamento de repositório e à cultura do ambiente. Além disso, também foram identificados como resultados que a correção de falhas do *build* é postergada e que os principais efeitos negativos das más práticas estão relacionados ao gerenciamento de projetos, bem como ao desenvolvimento e à infraestrutura de IC.

5 INVESTIGAÇÃO EMPÍRICA DA INFLUÊNCIA DAS MÁS PRÁTICAS DE INTEGRAÇÃO CONTÍNUA NA QUALIDADE DO SOFTWARE

Neste capítulo, é apresentado um estudo quantitativo para analisar o impacto das más práticas de IC na qualidade de software. Foi realizada uma análise quantitativa dos dados de nove projetos e das percepções de membros das equipes dos mesmos. O estudo foi publicado no 10º *Workshop* de Visualização, Evolução e Manutenção de Software (VEM) (Silva; Bezerra, 2022).

5.1 Introdução

A IC emerge como uma técnica crucial na engenharia de software desde os anos 70, experimentando uma adoção generalizada em empresas de desenvolvimento de software nas últimas décadas (Brooks, 1978; Beller *et al.*, 2017).

A utilização da IC no ciclo de desenvolvimento de sistemas promete vantagens consideráveis, incluindo a elevação na frequência de entrega de funcionalidades (Goodman; Elbaz, 2008; Vasilescu *et al.*, 2015), o aprimoramento da produtividade das equipes (Miller, 2008), a identificação precoce e correção de falhas (*bugs*) (Beller *et al.*, 2017), e o aperfeiçoamento na comunicação entre os membros das equipes (Downs *et al.*, 2010). Consequentemente, essa abordagem se estendeu amplamente a projetos comerciais e de código aberto, impulsionada por ferramentas que automatizam os processos e contribuem para os resultados positivos associados à IC (Felidré *et al.*, 2019).

Entretanto, implementar a IC em sistemas de média e alta complexidade não é tarefa simples (Shahin *et al.*, 2017). Requer um período de treinamento e adaptação de todos os membros do projeto para extrair os benefícios máximos da técnica e mitigar os riscos inerentes às más práticas de IC, que podem se manifestar se a implementação for inadequada (Ståhl *et al.*, 2017; Vassallo *et al.*, 2018). Estudos na literatura destacam os efeitos adversos das más práticas de IC em projetos de software (Duvall, 2018; Soares *et al.*, 2022). Zampetti *et al.* (2020) catalogaram 79 dessas más práticas, enquanto Elazhary *et al.* (2021) constataram, em seu estudo, uma variação significativa na implementação da IC, mesmo em empresas com modelos de negócios semelhantes.

Estudos recentes também investigam subpráticas de IC e seu impacto na qualidade de software. Santos *et al.* (2022) e Freitas *et al.* (2023) exploram o impacto das subpráticas e práticas de IC na qualidade do código e na produtividade de projetos de código aberto. Ambos os estudos

enfatizam que a adoção consistente e eficaz de subpráticas recomendadas, como maior cobertura de testes, redução da duração de compilação e aprimoramento da automação de testes, está associada a melhorias significativas na qualidade do código e na eficiência do desenvolvimento. Destacam que o simples uso de serviços de IC não garante resultados superiores; é essencial que os desenvolvedores sigam de forma consistente essas subpráticas para colher os benefícios da IC em seus projetos. Esses estudos ressaltam a importância de aderir a práticas recomendadas para alcançar melhorias na qualidade do código e na produtividade dos projetos de software.

Embora as subpráticas de IC demonstrem correlação positiva com a qualidade do código e a eficiência do desenvolvimento, é essencial reconhecer que a não adesão a essas práticas recomendadas pode resultar em efeitos prejudiciais. Sendo assim, destaca-se a importância de evitar as más práticas de IC, resguardando-se dos riscos que elas acarretam, em contraste com os benefícios observados quando as subpráticas de IC são seguidas de maneira consistente e eficaz.

Nesse contexto, este estudo se propõe a investigar o impacto das más práticas de IC na qualidade do software. Foram examinados nove projetos industriais de parceiros do setor, analisando dados quantitativos provenientes (1) dos sistemas selecionados, (2) da API do sistema de IC empregado nesses projetos e (3) das percepções das equipes de desenvolvimento envolvidas nos sistemas.

5.2 Projeto do estudo

5.2.1 Objetivo e questões de pesquisa

O principal objetivo no presente trabalho é analisar os efeitos das más práticas de IC em relação à qualidade do software. Assim, foram elaboradas as seguintes questões de pesquisa:

- **QP$_1$:** *Qual é o impacto nos atributos de qualidade interna de sistemas closed-source após a implantação da IC?* Por meio dessa questão de pesquisa, foi possível realizar uma análise inicial dos projetos selecionados para o estudo para verificar quaisquer diferenças nos atributos de qualidade interna que pudessem indicar a influência de más práticas no software analisado.
- **QP$_2$:** *O que os indicadores de qualidade ao longo do tempo podem revelar em projetos afetados por más práticas de IC?* Na segunda questão de pesquisa, investigou-se o comportamento dos indicadores de qualidade usados nos projetos analisados para verificar se a IC melhorou a qualidade do software ao longo do tempo.

– **QP$_3$:** *Quais más práticas de IC têm a maior prioridade de resolução para a melhoria da qualidade?* A última questão de pesquisa visa obter uma priorização das más práticas de IC com base no esforço para resolver e no impacto sobre a qualidade do software, de modo que a melhoria da qualidade possa ser alcançada de forma mais rápida e eficaz.

5.2.2 Passos do estudo

Passo 1: Selecionar sistemas para análise. Foram selecionados nove projetos de código fechado de parceiros da industria. A Tabela 12 resume os dados dos sistemas analisados. Na primeira coluna, é atribuído um identificador a cada sistema; na segunda coluna, apresenta-se uma breve descrição sobre a finalidade do software; e na coluna restante, o número total de pipelines de IC executados nos sistemas é apresentado. Os seguintes critérios de aceitação foram utilizados na escolha dos sistemas: (1) sistemas com pelo menos um ano desde o início de seu desenvolvimento e (2) sistemas que têm configuração de IC há pelo menos seis meses. No trabalho anterior apresentado no Capítulo 4, foram identificadas más práticas no ambiente do presente estudo, portanto, esse fator não foi incluído nos critérios de aceitação.

Tabela 12 – Dados gerais dos sistemas de software analisados.

SISTEMA	DOMÍNIO	PIPELINES
S1	Gerenciamento organizacional baseado em habilidades	1.764
S2	Assistência estudantil	731
S3	Proficiência leitora em idioma estrangeiro	2.135
S4	Gerenciamento de atividades complementares para graduação	505
S5	Gerenciamento de almoxarifado	106
S6	Gerenciamento de riscos organizacionais	818
S7	Gerenciamento de projetos sociais	303
S8	Prontuário odontológico eletrônico	1.218
S9	Gestão de eventos acadêmicos	255

Fonte: Elaborado pelo autor.

Passo 2: Coletar e analisar atributos internos de qualidade. Para responder à QP$_1$, foi realizada uma análise preliminar dos projetos separando aqueles que não adotaram a IC desde seu início totalizando cinco projetos. Os valores dos atributos de qualidade interna dos sistemas selecionados foram coletados em dois momentos, um antes e outro depois da introdução do IC no projeto. As métricas foram coletadas por meio de uma licença não comercial da ferramenta Understand[1]. A análise de atributos foi realizada comparando os valores das métricas antes e depois da introdução da IC nos projetos.

[1] https://www.scitools.com/

Passo 3: Coletar e analisar o histórico dos indicadores de qualidade. Para complementar a análise dos atributos internos de qualidade e responder à nossa QP_2, foi coletado o histórico dos indicadores de qualidade de todos os sistemas selecionados. Os indicadores de qualidade observados foram (1) Confiabilidade do software, (2) Segurança do software e (3) Manutenibilidade do software. A coleta dos indicadores foi realizada por meio de um *script* desenvolvido para esse fim[2], de forma que foi possível obter os valores dos indicadores de qualidade automaticamente a partir do sistema de IC utilizado nos projetos. Especialistas convidados validaram o roteiro de coleta e os dados obtidos. Além dos indicadores de qualidade, foi coletado o histórico de problemas levantados pela análise estática do projeto. A análise dos indicadores de qualidade foi realizada comparando os valores dos indicadores em cada um dos projetos.

Passo 4: Desenvolver, aplicar e analisar o questionário . Para responder à última questão de pesquisa, foi desenvolvido um questionário[3] destinado aos membros das equipes de desenvolvimento que trabalharam nos projetos analisados. O formulário era composto por duas perguntas objetivas com o objetivo de (1) descobrir as relações entre as más práticas de IC e os indicadores de qualidade e (2) entender o nível de esforço necessário para corrigir cada má prática nos projetos. Foi realizada uma aplicação inicial do questionário com um especialista para fins de validação. Com base no *feedback* obtido, foi possível refatorar alguns pontos. Foi criado um material explicativo (que foi referenciado no questionário) sobre más práticas de IC e indicadores de qualidade para auxiliar os participantes a responder o questionário. Também foram coletados dados de caracterização dos participantes: o nível de estudo, a função desempenhada no projeto e o conhecimento sobre IC e qualidade de software. No total, foram obtidas 21 respostas (de 380 convites enviados) de pessoas que faziam parte da equipe de desenvolvimento dos sistemas analisados. A Tabela 13 apresenta as más práticas de IC consideradas no presente estudo. As informações presentes na tabela são o identificador de cada má prática, uma breve descrição e a categoria a qual ela está associada [4]. Este estudo aproveitou os resultados do estudo anterior descrito no Capítulo 4 o qual buscou identificar a existência das más práticas de IC no mesmo ambiente do presente estudo.

As respostas foram analisadas da seguinte forma: primeiro, observou-se a relação entre as más práticas de IC e os indicadores de qualidade para classificar (1) as más práticas de acordo com seu nível de impacto e (2) os indicadores de qualidade pelo nível de impacto sofrido.

[2] Os artefatos estão disponíveis em: https://github.com/ruben-silva-dev/dissertacao-pcomp-ic?tab=readme-ov-file#artefatos-do-segundo-estudo

[3] Ver o Apêndice B.

[4] As categorias obedecem a mesma nomenclatura estabelecida na Tabela 7.

Tabela 13 – Más práticas de IC analisadas

ID	MÁ PRÁTICA	CATEGORIA
MP1	Algumas tarefas de pipelines são iniciadas manualmente	OPB
MP2	*Feature branches* são usadas em vez de *feature toggles*	REP
MP3	Falta de teste em um ambiente de produção	GQ
MP4	*Branches* divergentes	REP
MP5	Ferramentas externas são usadas com suas configurações padrão	INF
MP6	Desenvolvedores e operadores são mantidos como funções separadas	CUL
MP7	Notificações de falhas são enviadas apenas para equipes/desenvolvedores alocados na tarefa	OPB
MP8	Um *build* falha devido a alguma instabilidade na execução que não deveria ocorrer	OPB
MP9	*Quality gates* são definidos sem os desenvolvedores, considerando apenas o que é ditado pelo cliente	GQ
MP10	O hardware do servidor IC é usado para finalidades diferentes, além da execução da estrutura de IC	INF
MP11	Os desenvolvedores não têm controle total do ambiente	CUL
MP12	Scripts de *build* longos	MB
MP13	Os casos de teste não são organizados em pastas com base em seus propósitos	REP
MP14	Testes ausentes em *feature branches*	GQ

Fonte: Elaborado pelo autor.

Por fim, as más práticas de IC foram classificadas de acordo com o esforço necessário para resolvê-las. Para isso, foi usada uma estratégia de pesos associados às opções disponíveis para o nível de esforço de resolução: peso 1 para o nível baixo, peso 2 para o nível médio e peso 3 para o nível alto. Foi calculada uma pontuação para cada má prática multiplicando o número de ocorrências de cada nível de impacto por seu respectivo peso. Por fim, foi feita a correlação entre o nível de impacto na qualidade e o esforço para resolver cada má prática. Uma lista priorizada de más práticas foi gerada e traduzida em uma matriz de esforço versus impacto que pode ser útil na seleção de más práticas prioritárias para resolução, com o objetivo de melhorar a qualidade do software.

5.3 Resultados

5.3.1 Impacto nos atributos internos de qualidade após a introdução da IC (QP_1)

Para responder à QP_1, foi realizada a análise estática do código-fonte dos cinco projetos pré-selecionados que não usaram a IC desde o início para obter os valores dos atributos de qualidade interna. Para isso, considerou-se duas versões do código de cada sistema: a primeira foi baseada no ponto imediatamente anterior à implementação da IC no projeto. Como resultado, foi possível observar um aumento nos valores brutos de todos os atributos internos de qualidade para cada um dos cinco sistemas. A tabela 14 mostra os resultados dos dados coletados para cada sistema.

A análise partiu da premissa de que o atributo Coesão é inversamente proporcional aos outros atributos (ou seja, quanto maior o valor, melhor). Assim, foi possível observar melhorias significativas na coesão de todos os sistemas, especialmente nos sistemas S1 e S4.

Tabela 14 – Impacto da implantação de IC nos atributos internos de qualidade dos sistemas.

Sistema	Coesão	Complexidade	Herança	Acoplamento	Tamanho
S1	2290 ⇒ 3788 (↑ 65.41%)	718 ⇒ 955 (↑ 33%)	178 ⇒ 230 (↑ 29.21%)	172 ⇒ 290 (↑ 68.6%)	4008 ⇒ 5621 (↑ 40.24%)
S2	1399 ⇒ 1557 (↑ 11.29%)	331 ⇒ 370 (↑ 11.78%)	85 ⇒ 143 (↑ 68.23%)	46 ⇒ 73 (↑ 58.69%)	1670 ⇒ 2101 (↑ 25.8%)
S4	1247 ⇒ 1717 (↑ 37.69%)	355 ⇒ 497 (↑ 40%)	111 ⇒ 129 (↑ 16.21%)	69 ⇒ 102 (↑ 47.82%)	1774 ⇒ 2576 (↑ 45.20%)
S6	2608 ⇒ 2668 (↑ 2.3%)	886 ⇒ 1044 (↑ 17.83%)	257 ⇒ 258 (↑ 0.38%)	180 ⇒ 191 (↑ 6.11%)	5068 ⇒ 5729 (↑ 13.04%)
S8	4478 ⇒ 4846 (↑ 8.21%)	1559 ⇒ 1859 (↑ 19.24%)	353 ⇒ 390 (↑ 10.48%)	553 ⇒ 611 (↑ 10.48%)	8863 ⇒ 11541 (↑ 30.21%)

Fonte: Elaborado pelo autor.

Como visto na Tabela 14, os valores de todos os atributos de qualidade interna aumentaram depois que os parceiros adotaram a IC nos projetos. É importante observar que é esperado o aumento dos atributos internos, como a complexidade, durante o desenvolvimento. No entanto, outro fato essencial a ser mencionado é que a coesão do código também aumentou em todos os casos, o que é um resultado significativo. Isso sugere que a adoção da IC pode ter contribuído para a manutenção e até mesmo aprimoramento da coesão do código.

Esses resultados refletem a importância da integração contínua não apenas na estabilidade e eficiência dos projetos, mas também na qualidade interna do código. A coesão do código é fundamental para a manutenção, compreensão e evolução do software ao longo do tempo, e seu aumento demonstra que a IC pode influenciar positivamente aspectos-chave da qualidade do código.

Uma possível explicação para a melhoria da coesão do código é a implementação da análise estática como parte do processo de IC. A análise estática é um processo vital para identificar problemas de qualidade de código, tais como redundância, complexidade desnecessária e baixa coesão. Ao integrar a análise estática em cada etapa do ciclo de desenvolvimento, os desenvolvedores podem identificar e corrigir problemas de código mais rapidamente, resultando em um código mais coeso e de melhor qualidade.

Além disso, a integração contínua promove uma cultura de revisão e refatoração contínuas. Com *builds* automatizadas e *feedback* rápido, os desenvolvedores são incentivados a realizar melhorias incrementais no código constantemente. Isso pode levar a uma redução na acumulação de dívidas técnicas e a uma maior qualidade geral do software.

Outro aspecto importante a ser considerado é o impacto da integração contínua na colaboração e comunicação dentro da equipe. A implementação de práticas de IC promove uma abordagem colaborativa para o desenvolvimento de software, onde os desenvolvedores

estão constantemente integrando e compartilhando seu trabalho. Isso pode levar a uma maior conscientização sobre a qualidade do código e a uma mentalidade coletiva de responsabilidade pela qualidade do produto final.

Em resumo, os resultados indicam que a adoção da IC não só melhora a estabilidade e eficiência dos projetos, mas também pode ter um impacto positivo na qualidade interna do código. A coesão do código aumentou em todos os casos, sugerindo que a IC pode contribuir para a manutenção e aprimoramento da qualidade do código ao longo do tempo.

Descoberta 5 *O desenvolvimento auxiliado pela IC pode ajudar a aumentar o grau de coesão dos sistemas.*

5.3.2 Análise da evolução dos indicadores de qualidade (QP_2)

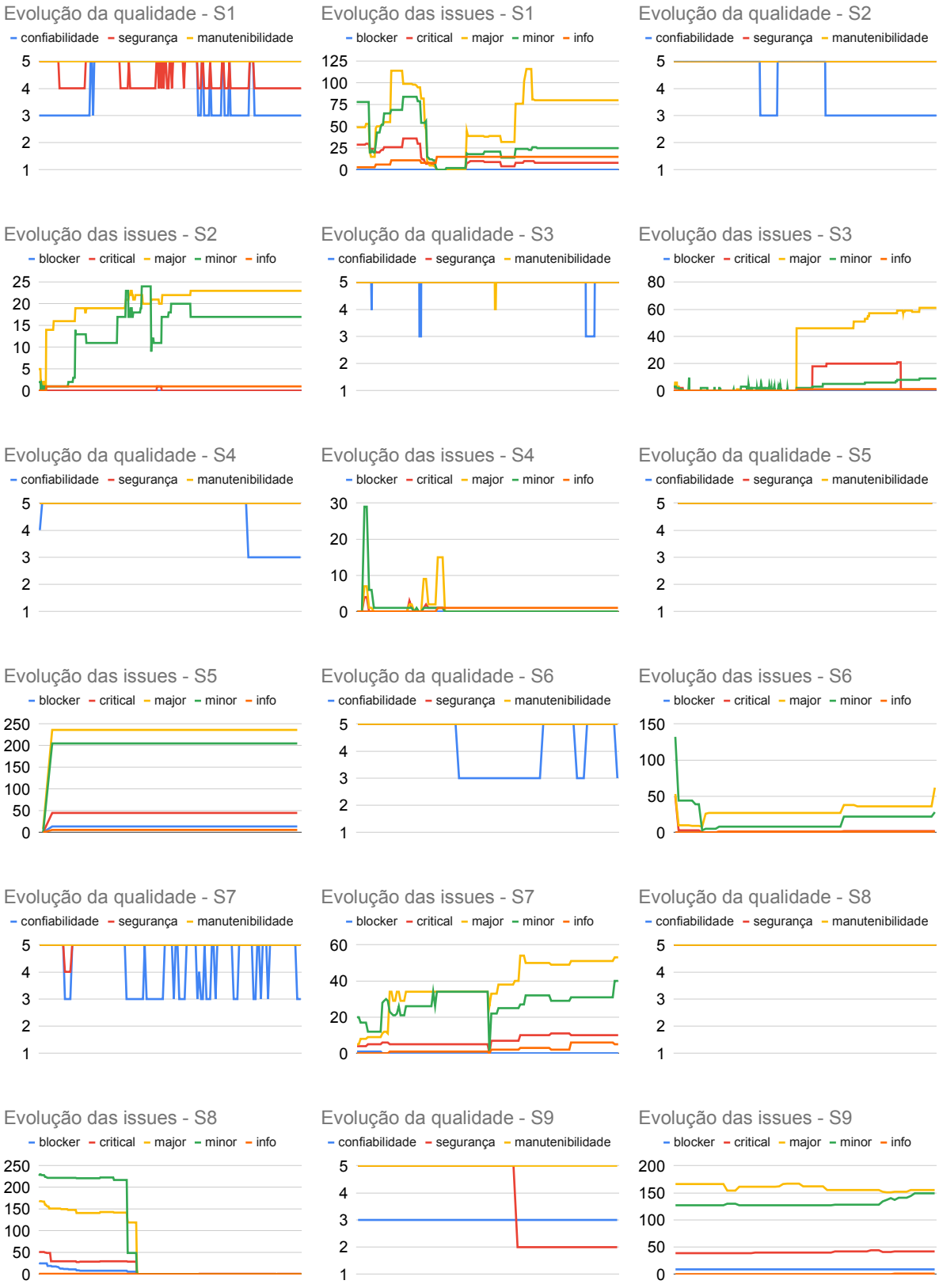
Foi realizada uma análise histórica dos indicadores de qualidade para todos os sistemas selecionados. A partir dos dados brutos coletados no sistema de IC dos projetos, foram gerados gráficos para representar o comportamento de (1) os valores dos indicadores de qualidade e (2) o número de problemas relatados pelo sistema de análise de código estático usado no ambiente de estudo (nesse caso, *SonarQube*[5]). Os gráficos resultantes foram compilados na Figura 4. Os gráficos representam a quantidade real de problemas relatados pelo sistema de análise estática. Como pode ser observado, os indicadores de qualidade estão associados a uma escala que varia de 1 a 5, em que um valor mais alto implica uma melhor qualidade para o sistema. No caso das *issues*, os valores foram divididos em cinco categorias de acordo com a gravidade de cada problema (*blocker*, *critical*, *major*, *minor*, *info*).

Como pode ser observado nos gráficos de cada sistema, há uma variedade de comportamentos diferentes. Em geral, é possível notar uma relação entre o aumento no número de *issues* e uma diminuição nos indicadores de qualidade interna do código. Por exemplo, sistemas com um grande número de *issues* tendem a apresentar maior complexidade e duplicação de código, o que pode afetar negativamente a manutenibilidade e a estabilidade do software.

Além disso, é interessante observar que, com exceção do sistema S8, todos os sistemas mantêm problemas não resolvidos ao longo do tempo. Isso sugere que o processo de integração contínua, especificamente a análise estática, não está sendo utilizado adequadamente

[5] https://www.sonarqube.org/

Figura 4 – Indicadores de qualidade e histórico das *issues*



Fonte: elaborado pelo Autor.

em toda a organização. Essa inconsistência na aplicação da IC pode ser atribuída a várias razões, como falta de compreensão sobre a importância da análise estática, falta de recursos dedicados à sua implementação e monitoramento, ou até mesmo resistência à mudança por parte da equipe de desenvolvimento.

A existência de más práticas de IC na organização corrobora essa observação. Por exemplo, a falta de uma estratégia clara para definir os níveis de qualidade dos projetos e o uso de ferramentas de configuração padrão que podem não refletir as necessidades específicas dos sistemas são indícios de uma implementação inadequada da IC. Esses problemas indicam uma falta de alinhamento entre as práticas de IC e as necessidades reais dos projetos de software, o que pode levar a uma deterioração da qualidade do código ao longo do tempo.

Portanto, é possível afirmar que más práticas de IC podem prejudicar (mesmo que indiretamente) o nível dos indicadores de qualidade de um sistema de software. É essencial que as organizações invistam na adoção de práticas de IC eficazes e na criação de uma cultura que valorize a qualidade do código e a melhoria contínua do processo de desenvolvimento.

Além disso, é recomendável que as organizações revisem suas políticas e processos relacionados à integração contínua, garantindo que todos os membros da equipe compreendam a importância da análise estática e recebam o suporte e os recursos necessários para implementá-la corretamente. A conscientização sobre a importância da qualidade do código e o comprometimento com a adoção de boas práticas de IC são fundamentais para o sucesso a longo prazo de qualquer projeto de software.

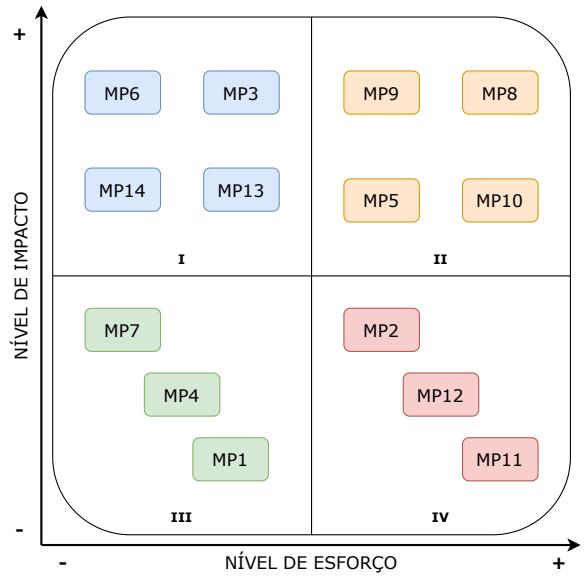
Descoberta 6 *As más práticas de IC podem prejudicar indiretamente o nível de qualidade de um produto de software.*

5.3.3 Priorização da resolução de más práticas de IC (RQ_3)

Para finalizar o estudo do impacto das más práticas de IC na qualidade do software no ambiente do estudo, foi realizada uma análise de correlação entre o nível de impacto e o nível de esforço para resolver as más práticas. Uma visão priorizada das más práticas de IC é apresentada na Figura 5. Como pode ser visto, a matriz é dividida em quadrantes que representam a prioridade de resolução das más práticas com base no nível de impacto na qualidade e no nível

de esforço necessário para resolvê-las. Assim, as más práticas no primeiro quadrante têm a maior prioridade de resolução, pois afetam significativamente a qualidade e não exigem muito esforço para serem corrigidas. Em seguida, vem o segundo quadrante com as más práticas que também têm um alto impacto, mas exigem muito esforço. Por fim, os dois últimos quadrantes contêm as más práticas de IC com baixo impacto na qualidade, mas com um nível de esforço suave e alto (3º e 4º, respectivamente).

Figura 5 – Matriz de nível de dificuldade e esforço



Fonte: Elaborado pelo Autor.

Por meio da análise da Figura 5, é possível verificar que as más práticas com maior prioridade estão relacionadas às categorias de Cultura (MP6), Garantia de Qualidade (MP3, MP14) e Repositório (MP13). Essas más práticas têm um impacto significativo na qualidade do software e exigem atenção especial por parte da equipe de desenvolvimento.

Por exemplo, a má prática MP6, relacionada à cultura de desenvolvimento, pode resultar em falta de comunicação e colaboração entre os membros da equipe, levando a problemas de integração e qualidade do código. Da mesma forma, as más práticas MP3 e MP14, que estão relacionadas à garantia de qualidade, podem resultar em testes inadequados ou insuficientes, aumentando o risco de *bugs* e falhas no software.

Além disso, a má prática MP13, relacionada ao repositório de código, pode levar a problemas de versionamento, dificultando o rastreamento de mudanças e a colaboração entre os desenvolvedores. Uma gestão inadequada do repositório também pode resultar em conflitos de código e perda de dados importantes.

Com base na descrição das más práticas, é possível dizer que é vital implementar um processo de teste bem estruturado e adotar uma cultura orientada a *DevOps* para obter melhores resultados na melhoria da qualidade do software. Isso inclui a integração contínua de testes automatizados, a revisão regular do código, a colaboração entre equipes de desenvolvimento e operações, e a automação de processos de construção, testes e implantação.

Além disso, é importante investir em ferramentas e tecnologias que possam facilitar a identificação e correção de más práticas, como ferramentas de análise estática de código, sistemas de monitoramento de qualidade do código e plataformas de integração contínua e entrega contínua (CI/CD).

Em resumo, a identificação e correção de más práticas de desenvolvimento são essenciais para garantir a qualidade e a estabilidade do software. Ao priorizar áreas críticas, como cultura, garantia de qualidade e gerenciamento de repositório, as equipes de desenvolvimento podem melhorar significativamente a qualidade do software e a satisfação dos usuários finais.

Descoberta 7 *Uma boa arquitetura de teste e a implementação de DevOps são práticas recomendadas relacionadas à IC que podem melhorar a qualidade do software.*

5.4 Ameaças à validade

Quanto às ameaças à validade do estudo, destacam-se: (1) o número de sistemas e membros da equipe considerados no estudo, o que pode ser explicado pela natureza de código fechado dos projetos; no entanto, buscou-se não limitar os dados coletados tanto da API do sistema de IC quanto do questionário que foi direcionado até mesmo para ex-membros da organização; (2) a confiabilidade das ferramentas de coleta de dados, que foi tratada por meio do uso de uma ferramenta consolidada para a coleta dos atributos internos de qualidade e da validação por especialistas do *script* e do questionário desenvolvido; e (3) a generalização dos resultados, uma vez que o estudo lidou com um cenário específico, assim, pode-se considerar que os resultados obtidos aqui são aplicáveis somente no contexto de pequenas organizações, com pouca maturidade de IC e que são afetadas por más práticas de IC.

5.5 Considerações finais

No presente trabalho, foi realizado um estudo empírico para investigar os efeitos das más práticas de IC em relação à qualidade do software. O estudo foi aplicado em uma pequena organização industrial caracterizada pela presença de más práticas de IC em seu processo de desenvolvimento e pela imaturidade em relação às práticas de IC.

Os resultados indicam que a inclusão da IC no processo de desenvolvimento de software pode trazer benefícios significativos. Em primeiro lugar, observou-se que a integração contínua pode ajudar a manter a coesão dos sistemas. Ao automatizar o processo de integração e implementar práticas de revisão de código e análise estática, os desenvolvedores podem identificar e corrigir problemas de integração e qualidade do código de forma mais rápida e eficiente.

No entanto, também foi possível notar que as más práticas de IC podem, mesmo que indiretamente, prejudicar a qualidade do software. A falta de uma cultura de desenvolvimento colaborativa, testes inadequados e problemas de gestão de repositório são exemplos de más práticas que podem impactar negativamente a qualidade do código e a estabilidade do sistema.

Uma das principais conclusões do estudo é que uma boa arquitetura de teste e a implementação de uma cultura *DevOps* podem contribuir significativamente para a melhoria da qualidade do software. Isso inclui a automação de testes, a integração contínua de testes automatizados e a colaboração entre equipes de desenvolvimento e operações para garantir uma entrega contínua e confiável de software.

Além disso, recomenda-se que as organizações invistam na formação e conscientização dos membros da equipe sobre as boas práticas de IC e adotem ferramentas e tecnologias que facilitem a identificação e correção de problemas de integração e qualidade do código.

Em resumo, o estudo destaca a importância da integração contínua e da adoção de práticas de IC eficazes para garantir a qualidade e a estabilidade do software. Ao priorizar uma cultura de desenvolvimento colaborativa, uma arquitetura de teste robusta e a automação de processos de integração e entrega, as organizações podem melhorar significativamente a qualidade do seu software e a satisfação dos seus clientes.

6 REVELANDO A RELAÇÃO ENTRE INTEGRAÇÃO CONTÍNUA E REVISÃO DE CÓDIGO: UM ESTUDO COM 10 PROJETOS CLOSED-SOURCE

6.1 Introdução

No desenvolvimento de software atual, duas práticas essenciais amplamente conhecidas são adotadas tanto por projetos industriais quanto por projetos de código aberto para alavancar seus processos de desenvolvimento (Tsotsis, 2011; Zampetti *et al.*, 2019; Wessel *et al.*, 2020; Zampetti *et al.*, 2017; Meyer, 2014). A primeira é a RC, que tem como objetivo detectar e corrigir problemas introduzidos durante o desenvolvimento (Bacchelli; Bird, 2013; Uchôa *et al.*, 2020; McIntosh *et al.*, 2016; Thongtanunam *et al.*, 2016). A RC é geralmente realizada por dois ou mais desenvolvedores (revisores de código). Eles identificam o problema ao analisar possíveis questões que afetam uma alteração de código desenvolvida por outro programador. Os revisores então relatam uma lista de problemas identificados ao autor da alteração de código para correção (Bacchelli; Bird, 2013; Rigby; Bird, 2013). A segunda prática é a IC, que estabelece um conjunto de procedimentos que possibilita a integração frequente de código pelos desenvolvedores na base de código (Hilton *et al.*, 2016; Laukkanen *et al.*, 2015; Fitzgerald; Stol, 2017). Alguns ferramentas suportam ambas as práticas; por exemplo, para RC, há o GitLab e Gerrit, e para IC, existem o Travis e GitLab CI.

Apesar de ambas IC e RC serem práticas amplamente utilizadas, elas enfrentam desafios identificados ao longo do tempo. Em relação à IC, podem ser destacados: (1) estratégia deficiente de gerenciamento de dependências (o que pode levar a um aumento no tempo de compilação) (Jaspan *et al.*, 2018); (2) esquema de compilação muito complexo (Laukkanen *et al.*, 2017; Duvall, 2018); (3) estratégias de teste inadequadas (Shahin *et al.*, 2017; Debbiche *et al.*, 2014); e (4) *commits* muito longos ou *branches* de trabalho com ciclo de vida longo (isso pode levar a conflitos de código) (Laukkanen *et al.*, 2017; Duvall, 2018; Elazhary *et al.*, 2021). Em relação à RC, existem dificuldades tanto relacionadas aos desenvolvedores quanto aos revisores de código, entre as quais podemos destacar: (1) atraso na realização das revisões (que pode ser causado pela falta de priorização e acumulação de atividades para revisão) (Jiang *et al.*, 2021; Saini; Britto, 2021); (2) documentação insuficiente de mudanças ou mudanças substanciais (o que pode levar a problemas de interpretação no momento da revisão); e (3) feedback de revisão insuficiente ou não armazenado corretamente (por exemplo, feedback oral) (MacLeod *et al.*, 2018; Fatima *et al.*, 2018).

Apesar dos desafios mencionados, não muitos estudos investigaram a relação entre IC e RC (Rahman; Roy, 2017; Zampetti *et al.*, 2019; Wessel *et al.*, 2020; Cassee *et al.*, 2020). Os estudos estavam relacionados a: (1) impacto da IC na RC do ponto de vista de fatores não técnicos, em como a carga de trabalho do revisor, participação do desenvolvedor e até mesmo o tamanho do patch afetam o processo de CR (Cassee *et al.*, 2020); (2) interação na discussão do *merge request* e o uso da IC dentro das discussões dos *merge requests* (Zampetti *et al.*, 2019); (3) expectativas e percepção de mudanças provocadas por bots de revisão de código (Wessel *et al.*, 2020); e (4) como as discussões de revisão de *merge requests* mudaram após a introdução da IC (Cassee *et al.*, 2020). Além disso, grande parte dos estudos de RC e IC utiliza projetos de código aberto (Zampetti *et al.*, 2019; Cassee *et al.*, 2020; Wessel *et al.*, 2020; Cassee *et al.*, 2020). Poucos estudos exploram as práticas de RC e IC em projetos industriais (de código fechado) (Zampetti *et al.*, 2019; Elazhary *et al.*, 2021).

Neste contexto, o presente estudo investiga a relação entre os processos de IC e RC para encontrar correlações entre os dois processos e como os revisores de código percebem a relação entre IC e RC em projetos de software. Além disso, como vários trabalhos relatam a existência de más práticas de IC em projetos (Hilton *et al.*, 2017; Olsson *et al.*, 2012; Ghaleb *et al.*, 2019), o estudo verifica se as más práticas de IC podem prejudicar o processo de RC em projetos. Por fim, é importante saber os benefícios e possíveis desafios que os revisores de código percebem em relação ao uso de IC e RC em projetos de software.

Para abordar essas lacunas, foi realizado um estudo quantitativo/qualitativo para explorar a relação entre IC e RC em dez projetos de código fechado desenvolvidos por parceiros industriais. Foi considerado um total de 32 métricas que foram coletadas e analisadas. Para completar as análises foram usadas as percepções das equipes responsáveis pelos projetos quanto a (1) validação das correlações encontradas na análise quantitativa, (2) efeitos das más práticas de IC na RC e (3) benefícios/desafios do uso conjunto de IC e RC em projetos.

Como resultado, foi possível identificar 23 correlações entre os processos de IC e RC nos projetos analisados, identificando as seguintes descobertas: (1) a carga de trabalho da IC bem como seu tempo de execução podem influenciar o tempo de revisão do código, (2) os tempos de execução da IC e da RC são mais correlacionados quando os dois processos ocorrem sequencialmente, (3) as más práticas em IC impactam o processo de RC com atrasos, aumento da carga de trabalho e maior chance de problemas não identificados, (4) a Integração Contínua na Revisão de Código agrega valor ao antecipar problemas, facilitar a distribuição de atualizações

e reduzir tarefas manuais, fortalecendo a eficiência do ciclo de desenvolvimento de software e (5) a utilização conjunta da IC e RC apresenta desafios, como garantir a qualidade do código, resolver conflitos e harmonizar processos. Isso demanda abordagens estratégicas e adaptativas para otimizar o ciclo de desenvolvimento de software.

6.2 Projeto do estudo

6.2.1 Objetivo e questões de pesquisa

Este estudo teve como objetivo analisar a relação entre os processos de IC e RC. As métricas de IC e RC e as percepções das equipes de projetos industriais apoiam a investigação. As percepções das equipes serviram de base para identificar (1) a visão dos membros sobre as correlações entre os processos de IC e RC, (2) o impacto das más práticas de IC em relação ao processo de RC e (3) os benefícios e desafios do uso conjunto dos processos de IC e RC. Como esta é uma pesquisa quantitativa/qualitativa, foi considerado apenas o contexto de projetos de código fechado. Cada questão de pesquisa (QP) é detalhada a seguir.

QP_1: *Quais aspectos do processo de IC e RC estão correlacionados?* A **QP_1** tem como objetivo verificar as correlações existentes entre os vários aspectos relacionados aos processos de IC e RC em projetos industriais. Por meio dessa QP, espera-se entender se essas duas atividades de engenharia de software são descritas na prática. As correlações investigadas aqui servem como base para a **QP_2**.

QP_2: *Quais são as percepções das equipes de desenvolvimento sobre a correlação entre os processos de IC e RC?* A **QP_2** tem como objetivo investigar as percepções das equipes de desenvolvimento dos projetos analisados sobre as correlações identificadas na **QP_1**. Assim, ao explorar a percepção dos desenvolvedores sobre as correlações, é possível validar as correlações identificadas e capturar detalhes implícitos dos processos de IC e RC dos parceiros industriais. Além disso, ao responder à **QP_2**, existe a possibilidade de se revelar possíveis falhas nos processos de IC e RC que podem ter influenciado as correlações existentes.

QP_3: *Como as más práticas de IC afetam a RC?* A **QP_3** tem como objetivo identificar a percepção das equipes de desenvolvimento dos projetos analisados em relação ao impacto das más práticas de IC no processo de RC. As más práticas foram identificadas por um estudo anterior no contexto dos parceiros industriais (Silva; Bezerra, 2020). Assim, pretende-se descobrir se as más práticas de IC podem interferir em aspectos que vão além da eficiência do próprio processo

de IC.

QP$_4$: *Quais são os benefícios e desafios gerados pela IC no processo de RC?* A **QP$_4$** tem como objetivo investigar como os resultados gerados a partir da execução do processo de IC são usados na RC. Nesta QP, é realizada uma extensão da análise da relação entre IC e RC das questões de pesquisa anteriores. Especificamente, foi observado o comportamento dos revisores de código no contexto de projetos de software que adotam ambos os processos.

6.2.2 Caracterização da empresa e de seu processo de IC e RC

A organização estudada atua no ambiente de uma instituição de ensino superior e oferece oportunidades de estágio na área de tecnologia da informação para estudantes de graduação. A maior parte da força de trabalho é composta por estudantes prestes a se formar, que atuam como estagiários e realizam diversas atividades, entre elas, desenvolvimento, projeto e prototipagem de software, análise de requisitos e manutenção de softwares previamente desenvolvidos. O principal objetivo da organização é fornecer soluções de software para atender às demandas da comunidade acadêmica e de parceiros externos que buscam soluções de software para suas necessidades.

Em relação aos processos internos, a organização adota metodologias ágeis e conta com o SCRUM (Pham; Pham, 2011) para o desenvolvimento de software. Ela também implementa práticas de garantia de qualidade, como IC e RC. Com relação à IC, a organização usa testes automatizados e análise estática durante a implementação na máquina do desenvolvedor e na compilação automatizada da IC. Quanto à RC, essa é uma atividade realizada pelos líderes técnicos das equipes de desenvolvimento, que têm mais *know-how* para identificar problemas de implementação, complementando as análises automatizadas realizadas no processo de IC.

A equipe de supervisão é composta por 06 analistas de tecnologia da informação que coordenam e lideram os projetos. Já a equipe de desenvolvimento conta com estagiários de instituições de ensino superior, responsáveis por desenvolver e testar as funcionalidades projetadas para os sistemas.

A organização tem buscado aprimorar seus processos de desenvolvimento, incorporando práticas e ferramentas de engenharia de software amplamente utilizadas no setor. Isso inclui a adoção dos processos de IC e Entrega Contínua (EC), que são fundamentais para garantir a entrega segura e confiável das funcionalidades aos clientes e evitar erros de implementação no ambiente de produção (Singh *et al.*, 2019).

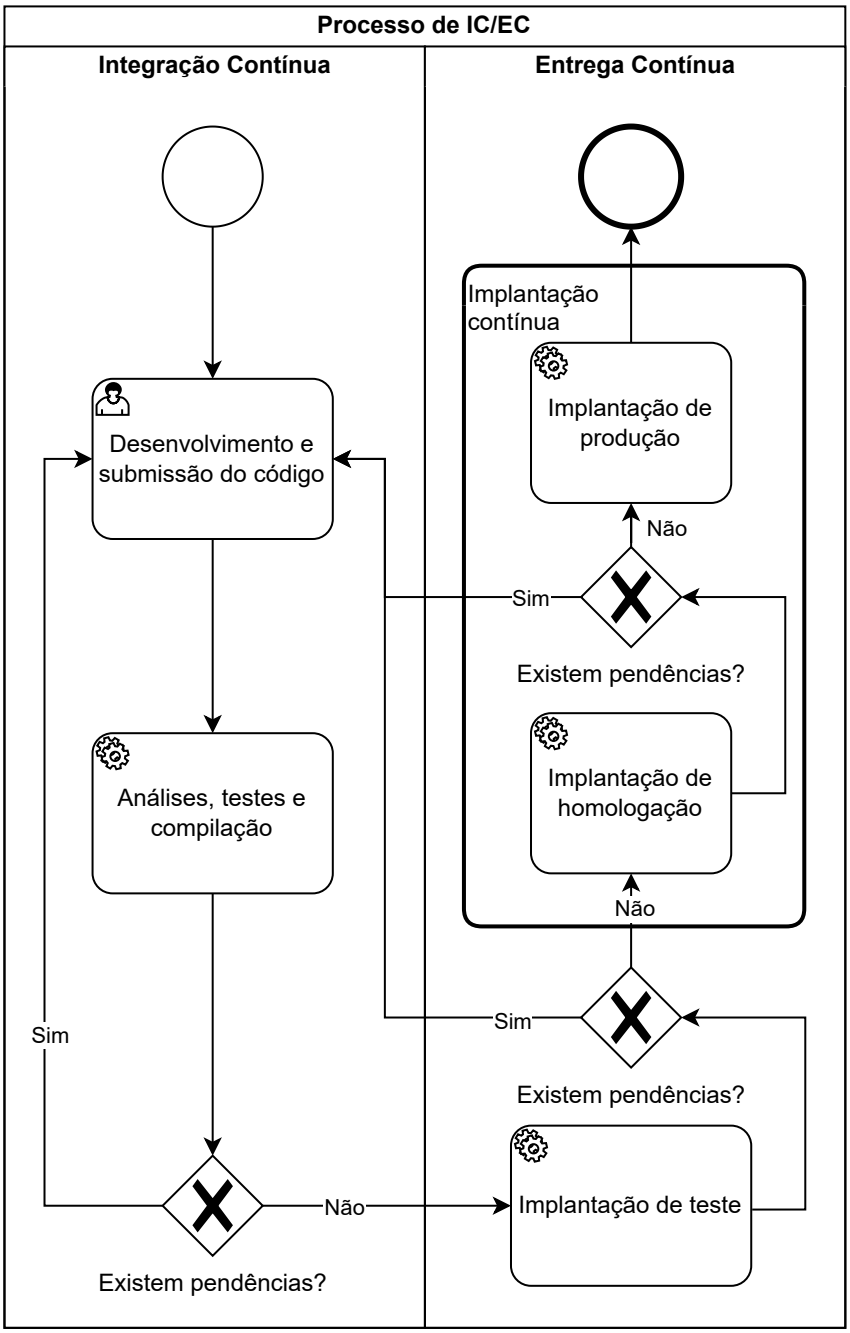
A Figura 6 mostra o processo de IC/EC adotado pela organização, que envolve a implementação de alterações no software, análise, teste, construção de código e implantação do sistema em diferentes ambientes, culminando com a implantação no ambiente de produção após a validação do cliente. O processo começa com o trabalho do desenvolvedor que é responsável pela implementação e submissão do código ao servidor de gerência de configuração, em seguida o servidor de IC detecta as novas mudanças e inicia as análises, testes e compilações do código-fonte para identificar problemas pendentes de resolução. Em caso de pendências o processo volta para a fase de desenvolvimento para correção dos problemas identificados pela IC, caso contrário o processo avança para a fase de entrega. A fase de entrega é responsável pela implantação da aplicação em diferentes ambientes, primeiramente no ambiente de testes onde os desenvolvedores poderão testar as novas *features*. Caso tudo ocorra bem o processo avança para a implantação de homologação para validação com o próprio cliente. Com tudo estável, a aplicação é implantada no ambiente de produção. Caso sejam detectadas pendências na implantação de teste e/ou de homologação o processo retorna para a fase de desenvolvimento.

Resumidamente, IC/EC significa Integração Contínua e Entrega/Implantação Contínua. É um método de entrega frequente de aplicações aos clientes por meio da introdução da automação nos estágios de desenvolvimento de uma aplicação. Os principais conceitos atribuídos à IC/EC são integração contínua, entrega contínua e implantação contínua. A integração contínua é um processo de automação para desenvolvedores que significa que novas alterações de código em um aplicativo são regularmente criadas, testadas e mescladas com segurança em um repositório compartilhado. A EC e a implantação contínua são conceitos relacionados que tratam da automação de outros estágios do pipeline, mas às vezes são usados separadamente para ilustrar a quantidade de automação que está ocorrendo. Com um pipeline de IC/EC, as equipes de desenvolvimento podem fazer alterações no código que são testadas automaticamente e enviadas para entrega e implantação. A IC/EC é importante porque ajuda as equipes de desenvolvimento, segurança e operações a trabalhar da forma mais eficiente e eficaz possível, diminuindo o trabalho de desenvolvimento manual, tedioso e demorado e os processos de aprovação herdados, liberando as equipes de *DevOps* para serem mais inovadoras no desenvolvimento de software. O *pipeline* de IC/EC faz parte da estrutura mais ampla de *DevOps*/*DevSecOps*.

6.2.3 Etapas e procedimentos do estudo

As etapas para apoiar a investigação são descritas a seguir.

Figura 6 – Processo de IC/EC da organização



Fonte: elaborado pelo Autor.

6.2.3.1 Seleção de sistemas industriais para análise

Dos 46 projetos desenvolvidos pelos parceiros da industria foram selecionamos dez deles de acordo com os seguintes critérios: (1) sistemas configurados para IC por pelo menos um ano e (2) sistemas em operação por pelo menos um ano. Esses critérios foram aplicados porque a análise em um período mais abrangente seria necessária para obter resultados consistentes. A Tabela 15 apresenta dados gerais por sistema selecionado. A primeira coluna nomeia o sistema[1].

[1] Seus nomes reais foram omitidos devido a restrições de propriedade intelectual.

O restante das colunas apresenta o domínio do sistema, o número de versões (NV), o número de linhas de código (LOC), o número de pipelines (NP) e o número de *merge requests* (NMR).

Tabela 15 – Dados gerais sobre os sistemas analisados.

Sistema	Domínio	NV	LOC	NP	NMR
S1	Gerenciamento organizacional baseado em habilidades	1	39.585	1.764	367
S2	Assistência estudantil	1	22.391	731	614
S3	Proficiência em leitura em idioma estrangeiro	20	18.098	2.135	224
S4	Gerenciamento de atividades complementares para graduação	1	3.209	505	85
S5	Gerenciamento de almoxarifado	4	22.045	106	290
S6	Gerenciamento de riscos organizacionais	1	20.336	818	154
S7	Gerenciamento de projetos sociais	1	17.583	303	107
S8	Prontuário odontológico eletrônico	7	65.636	1.218	443
S9	Gestão de eventos acadêmicos	12	26.724	255	381
S10	Mineração de dados abertos	2	4.092	364	147

Fonte: Elaborado pelo autor.

É importante ressaltar que todos os sistemas selecionados dos parceiros industriais adotam o processo de RC durante o processo de desenvolvimento. Outra característica essencial é que os parceiros começaram a usar IC em seus projetos há cerca de 3 anos, o que nos leva a crer que eles possuem alguma experiência com este processo.

6.2.3.2 Seleção de métricas de IC e RC para análise

O trabalho de Chen *et al.* (2019) foi usado como base para selecionar as métricas do presente estudo. Esse estudo resume um conjunto amplamente utilizado de métricas extraídas de estudos anteriores (Gousios; Zaidman, 2014; Gousios *et al.*, 2014; Tsay *et al.*, 2014; Yu *et al.*, 2015; Zhang *et al.*, 2014; Cassee *et al.*, 2020; Chen *et al.*, 2019). Além disso, foram selecionadas as métricas que atendem aos seguintes critérios: (1) as métricas devem ter uma relação direta com IC ou RC, e (2) as métricas devem ser coletáveis considerando o ambiente do GitLab. Considerando esses critérios, 24 métricas foram escolhidas: 19 relacionadas à RC e 5 associadas à IC. Além disso, para capturar mais detalhes dos processos de IC e RC, foram consideradas mais oito métricas relacionadas aos seguintes aspectos: erros relatados pela IC, duração da revisão e quantidade de trabalho realizado pela IC, totalizando 32 métricas.

A Tabela 16 apresenta uma visão geral das 32 métricas selecionadas agrupadas por categoria (IC e RC) e dimensão (tempo, participação e intensidade). As dimensões podem ajudar a medir detalhes específicos dos processos de IC e RC em projetos: a dimensão **tempo** permite entender a duração dos eventos relacionados ao processo em questão (IC ou RC), enquanto a dimensão **participação** pode ajudar a saber como os membros da equipe do projeto agem

nos processos de IC e RC e, por fim, a dimensão **intensidade** dá a percepção da carga de trabalho realizada durante os processos de IC e RC. Vale a pena mencionar que, entre as métricas pertencentes à categoria IC, não existe nenhuma relacionada à dimensão participação porque o processo de IC ocorre de forma automatizada e, portanto, sem o contato direto dos membros da equipe do projeto.

Tabela 16 – Caracterização de métricas.

CATEGORIA	DIMENSÃO	ID	MÉTRICA	REFERÊNCIA	JUSTIFICATIVA
Revisão de Código	Tempo	M1	Tempo de Revisão	(Gousios; Zaidman, 2014)	Essa métrica é importante porque mede o tempo total necessário para completar uma revisão de código. Reflete a eficiência do processo de revisão e pode ser usada para identificar gargalos ou áreas para melhoria.
		M2	Tempo Médio entre Comentários	(Yu *et al.*, 2015)	Essa métrica ajuda a entender o padrão de interação durante a revisão de código. Um curto intervalo de tempo entre comentários pode indicar comunicação rápida e eficiente entre revisores, enquanto um tempo médio longo pode sugerir atrasos ou falta de envolvimento dos participantes.
		M3	Tempo entre o Primeiro e o Último Comentário	Trabalho Atual	Essa métrica captura a duração total da revisão de código, desde o primeiro até o último comentário. Ela fornece uma visão geral do tempo total dos revisores analisando e discutindo um trecho específico de código.
	Participação	M4	Número de Participantes	(Gousios; Zaidman, 2014; Gousios *et al.*, 2014)	O número de participantes é uma métrica relevante para avaliar o envolvimento da equipe de desenvolvimento na revisão de código. Um número adequado de revisores pode melhorar a qualidade e diversidade de opiniões expressas durante o processo de revisão.
		M5	Comentários *Inline*	(Cassee *et al.*, 2020; Chen *et al.*, 2019)	Essa métrica mede o número de comentários inseridos diretamente no código-fonte durante a revisão. Muitos comentários *inline* podem indicar áreas problemáticas no código ou oportunidades de melhoria.
		M6	Comentários Gerais	(Cassee *et al.*, 2020; Chen *et al.*, 2019)	Essa métrica abrange comentários não inseridos diretamente no código-fonte, mas feitos em uma seção de discussão separada. Esses comentários podem incluir discussões mais amplas sobre funcionalidade, design ou outras considerações relevantes.
		M7	Comentários Efetivos	(Cassee *et al.*, 2020; Chen *et al.*, 2019)	Essa métrica se refere a comentários que impactam significativamente a qualidade final do código. Comentários que apontam erros, fornecem sugestões úteis ou destacam áreas problemáticas podem ser considerados comentários efetivos.
		M8	Total de Comentários	(Gousios; Zaidman, 2014; Gousios *et al.*, 2014; Tsay *et al.*, 2014; Yu *et al.*, 2015)	Essa métrica conta o total de comentários feitos durante a revisão de código. Avaliar o nível de discussão e interação dos revisores pode ser significativo.
		M9	Tempo da Primeira Resposta	(Yu *et al.*, 2015; Chen *et al.*, 2019)	Essa métrica mede o tempo decorrido desde o início da revisão de código até a primeira resposta ou comentário. Ajuda a avaliar a prontidão e capacidade dos revisores em fornecer *feedback* inicial no tempo adequado.
		M10	Número de Menções	(Zhang *et al.*, 2014)	Essa métrica conta o número de menções específicas a determinados revisores ou seções de código. Avaliar o envolvimento individual dos revisores ou a relevância de certas seções de código pode ser relevante.
	Intensidade	M11	*Commits* auto-fundidos	(Yu *et al.*, 2015)	Essa métrica rastreia o número de *commits* feitos por revisores que, por sua vez, também fundiram o código. Pode indicar um potencial conflito de interesses ou falta de revisão adequada.
		M12	Linhas de Código Fonte Adicionadas	(Gousios; Zaidman, 2014; Gousios *et al.*, 2014; Yu *et al.*, 2015; Chen *et al.*, 2019)	Essa métrica mede o número de linhas de código-fonte adicionadas durante o processo de desenvolvimento. Pode ajudar a avaliar o esforço de desenvolvimento necessário e seu impacto nas atividades de revisão.
		M13	Linhas de Código Fonte Removidas	(Gousios; Zaidman, 2014; Gousios *et al.*, 2014; Yu *et al.*, 2015; Chen *et al.*, 2019)	Essa métrica mede o número de linhas de código-fonte removidas durante o processo de desenvolvimento. A remoção de código pode indicar melhorias, simplificações ou correções, mas também pode introduzir problemas se não for revisada adequadamente.
		M14	Linhas de Código de Teste Adicionadas	(Gousios; Zaidman, 2014; Gousios *et al.*, 2014)	Essa métrica conta o número de linhas de código especificamente adicionadas para fins de teste. Testes adequados são uma parte essencial do desenvolvimento de software, e um aumento no código de teste pode impactar a complexidade da revisão e a qualidade geral do código.
		M15	Linhas de Código de Teste Removidas	(Gousios; Zaidman, 2014; Gousios *et al.*, 2014)	Essa métrica mede o número de linhas de código de teste removidas durante o processo de desenvolvimento. A remoção de código de teste pode indicar refatoração ou melhoria no código de produção, mas também pode representar um risco se não for revisada adequadamente para os efeitos da remoção.
		M16	Número de Revisões	(Gousios; Zaidman, 2014; Gousios *et al.*, 2014; Tsay *et al.*, 2014; Yu *et al.*, 2015; Chen *et al.*, 2019)	Essa métrica conta o número de revisões de código conduzidas em um período determinado. Fornece uma visão geral da atividade geral de revisão e pode ser usado para identificar tendências ou variações ao longo do tempo.
		M17	Arquivos Adicionados	(Gousios; Zaidman, 2014)	Essa métrica conta o número de arquivos adicionados durante o desenvolvimento. Um aumento no número de arquivos pode afetar a complexidade e o escopo da revisão necessária.
		M18	Arquivos Deletados	(Gousios; Zaidman, 2014)	Essa métrica mede o número de arquivos deletados durante o desenvolvimento. A remoção de arquivos pode impactar a integridade e estabilidade do código existente e exigir revisão cuidadosa.
		M19	Arquivos Modificados	(Gousios; Zaidman, 2014)	Essa métrica conta o número de arquivos modificados durante o desenvolvimento. Modificações podem afetar o comportamento do código existente e requerem revisão para garantir a não introdução de *bugs* ou regressões.
		M20	Total de Alterações	(Gousios; Zaidman, 2014; Gousios *et al.*, 2014; Tsay *et al.*, 2014)	Essa métrica mede o número de alterações (adicionadas, deletadas ou modificadas) feitas durante o desenvolvimento. Pode ser útil para avaliar o tamanho e a complexidade geral do trabalho realizado, bem como seu impacto nas atividades de revisão.
Integração Contínua	Intensidade	M21	Total de *builds*	Trabalho Atual	Essa métrica conta o número total de *builds* executados no processo de integração contínua. Avaliar a frequência de *builds* e o esforço computacional associado pode ser útil.
		M22	*Builds* com sucesso	Trabalho Atual	Essa métrica conta o número de *builds* bem-sucedidos durante a integração contínua. Compilações bem-sucedidas indicam que o código é compilável e adequado para implantação ou desenvolvimento adicional.
		M23	Builds com falha	Trabalho Atual	Essa métrica conta o número de *builds* com falha durante a integração contínua. Falhas na compilação indicam que o código não é compilável e requer investigação e correção.
		M24	Total de jobs	Trabalho Atual	Essa métrica conta o número de jobs executados no processo de integração contínua. Avaliar o esforço computacional necessário e seu impacto no fluxo de trabalho geral pode ser relevante.
		M25	Jobs com sucesso	Trabalho Atual	Essa métrica mede o número de jobs que tiveram sucesso no processo de integração contínua. Sucessos de jobs indicam estabilidade do código e eficácia das práticas de integração contínua.
		M26	Jobs com falha	Trabalho Atual	Essa métrica mede o número de jobs com falha durante a integração contínua. Falhas de jobs podem indicar problemas no código ou no processo de integração contínua que requerem revisão e correção.
		M27	Testes executados	(Shi *et al.*, 2019)	Essa métrica mede o número total de testes executados durante o processo de integração contínua. Pode ser relevante para avaliar a cobertura de testes e a qualidade geral do código.
		M28	Testes com falha	(Shi *et al.*, 2019)	Essa métrica conta o número de testes com falha durante a integração contínua. Falhas nos testes indicam que o código está com mau funcionamento e requer revisão e correção.
		M29	Testes com erro	(Shi *et al.*, 2019)	Essa métrica conta o número de testes que resultaram em erros durante o processo de integração contínua. Erros nos testes indicam problemas de lógica ou configuração que requerem investigação e correção.
		M30	Testes pulados	(Shi *et al.*, 2019)	Essa métrica mede o número de testes pulados ou não executados durante o processo de integração contínua. Testes pulados podem indicar problemas com a configuração do ambiente de teste ou outras dependências.
	Tempo	M31	Latência da IC	(Yu *et al.*, 2015; Chen *et al.*, 2019)	Essa métrica mede a latência ou tempo de resposta entre uma alteração de código e a conclusão do processo de integração contínua. Latência excessiva pode indicar problemas de desempenho ou gargalos que afetam o desenvolvimento e revisão de código.
		M32	Intervalos de correção de builds	Trabalho Atual	Essa métrica mede o tempo decorrido entre um *build* com falha e o próximo *build* bem-sucedido após a correção dos problemas. Essa métrica pode ser relevante para avaliar a eficiência na identificação e resolução de problemas durante o processo de integração contínua.

Fonte: Elaborado pelo autor.

6.2.3.3 Coleta e validação do cálculo das métricas de IC e RC para análise

Para automatizar a coleta das métricas alvo, foi desenvolvido um *script* em Python[2] utilizando a biblioteca Python-GitLab[3] de modo a facilitar a obtenção dos dados, já que os sistemas alvo estão hospedados na plataforma GitLab. O fluxo de trabalho do *script* pode ser descrito da seguinte maneira: (1) primeiro, ocorre a autenticação com a *API* do GitLab; (2) em seguida, os dados primários dos projetos selecionados são recuperados; (3) para cada projeto, é recuperado o conjunto de todos os *merge requests* dos projetos analisados; (4) para cada *merge request*, todas as métricas são calculadas; e, finalmente, (5) os dados são armazenados em formato *CSV*, gerando um arquivo por projeto.

2 https://github.com/ruben-silva-dev/dissertacao-pcomp-ic/blob/main/estudo-3/mining-tool.zip

3 https://python-gitlab.readthedocs.io/en/stable/

A coleta de dados através do *script* retornou um conjunto de dados de 942 *merge requests* extraídos dos dez projetos de código fechado hospedados no GitLab. A coleta de cada projeto ocorreu individualmente, gerando vários arquivos *CSV*.

Para promover a confiabilidade dos dados coletados, foi feita a validação dos valores das métricas. Um dos membros do estudo fez a validação manual para identificar anomalias nos dados calculados pelo *script*. Assim, foi necessário implementar ajustes no código do *script*.

6.2.3.4 Realização da análise de dados

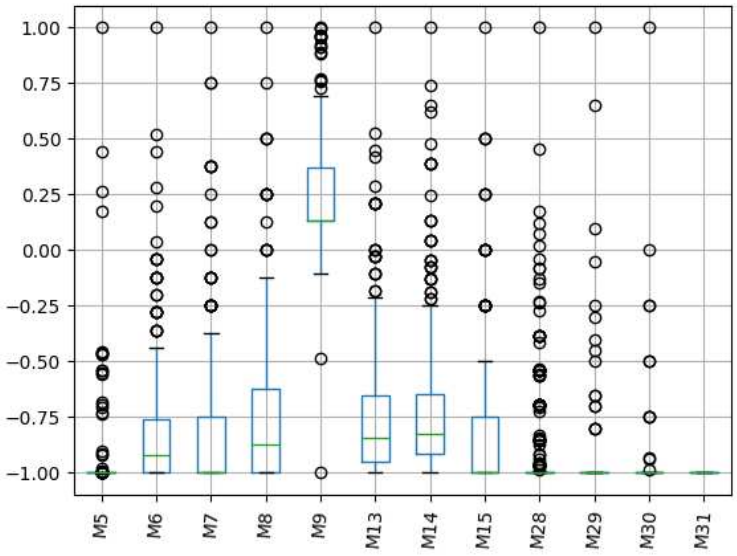
Para este propósito, foi aplicada uma análise estatística para identificar correlações entre os fatores considerados em nosso estudo. Neste estudo, foram analisados os dados obtidos no passo anterior sob duas perspectivas: de forma geral e separadamente em cada projeto. Para padronizar os valores de cada métrica, utilizou-se o procedimento de normalização de dados usando o método “MinMaxScaler” da biblioteca Python “sklearn” (Bisong, 2019). O método "MinMaxScaler"requer que os limites inferior e superior para a normalização sejam passados como parâmetros, então optamos por usar os valores -1 e 1, respectivamente.

Em seguida, foi realizada a análise de *outliers* para evitar viés em nosso conjunto de dados. A Figura 8 mostra o gráfico de *boxplot* geral para as métricas de RC, e a Figura 7 mostra o *boxplot* geral gerado a partir dos valores calculados para as métricas de IC. Foi possível observar muitos *outliers* em ambos os casos (RC e IC) que poderiam influenciar na análise das correlações. Portanto, foi preciso remover os *outliers* usando o teste de Tukey (Driscoll, 1996). No entanto, com a geração da matriz de correlação, não ocorreram alterações significativas nos valores dos coeficientes de correlação. Portanto, optou-se por considerar os *outliers* na análise das métricas uma vez que nosso *dataset* não continha uma quantidade significativa de dados.

Finalmente, para realizar a análise de correlação, primeiro foi analisada a distribuição dos dados considerando todas as métricas juntas, tanto IC quanto RC[4]. Ficou claro que os dados não seguem uma distribuição normal. Portanto, foi feito o cálculo do coeficiente de correlação de Spearman (Schober *et al.*, 2018) com um intervalo de confiança de 95% (valor $\rho \geq 0{,}05$). O coeficiente de correlação de Spearman varia de -1 a +1, onde 0 indica nenhuma correlação entre as variáveis envolvidas, e valores mais próximos de -1 ou +1 mostram uma correlação mais significativa entre as variáveis envolvidas (Schober *et al.*, 2018). Além disso, para entender

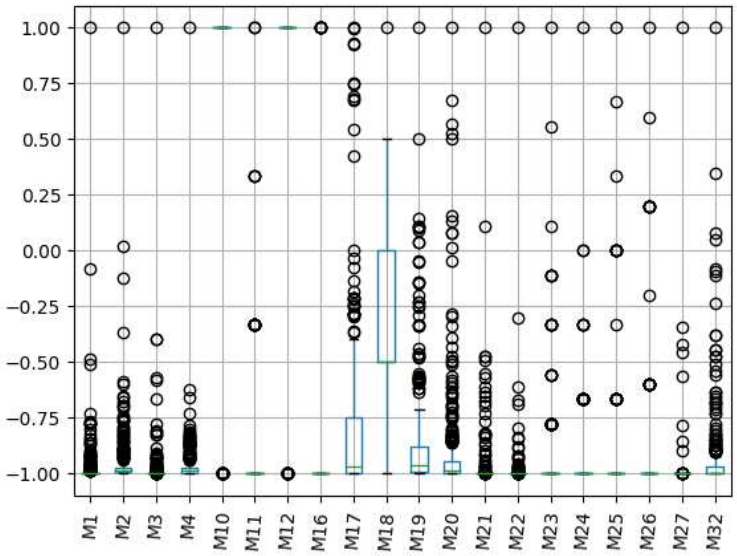
[4] Os gráficos contendo as distribuições estão disponíveis em https://github.com/ruben-silva-dev/dissertacao-pcomp-ic/tree/main?tab=readme-ov-file#artefatos-do-terceiro-estudo

Figura 7 – *Boxplot* geral das métricas de IC



Fonte: elaborado pelo Autor.

Figura 8 – *Boxplot* geral das métricas de RC



Fonte: elaborado pelo Autor.

melhor o coeficiente de correlação, foi usada a seguinte escala: (1) *correlação muito forte* (0,8 a 1,0 ou -0,8 a -1,0); (2) *correlação forte* (0,6 a 0,8 ou -0,6 a -0,8); (3) *correlação moderada* (0,4 a 0,6 ou -0,4 a -0,6); (4) *correlação fraca* (0,2 a 0,4 ou -0,2 a -0,4); e (5) *correlação muito fraca* ou *sem correlação* (0,0 a 2,0 ou 0,0 a -0,2). Por fim, as correlações significativas foram analisadas cuidadosamente no intuito de prover interpretações consistentes.

6.2.3.5 Grupo focal para coletar percepções dos gerentes

Para reunir as percepções dos especialistas que atuaram como revisores de código nos projetos analisados, foi utilizada uma dinâmica de grupo focal para abordar nossas últimas três questões de pesquisa. O planejamento do grupo focal foi realizado usando uma metodologia baseada no trabalho de Almeida *et al.* (2023). Os passos seguidos são descritos abaixo:

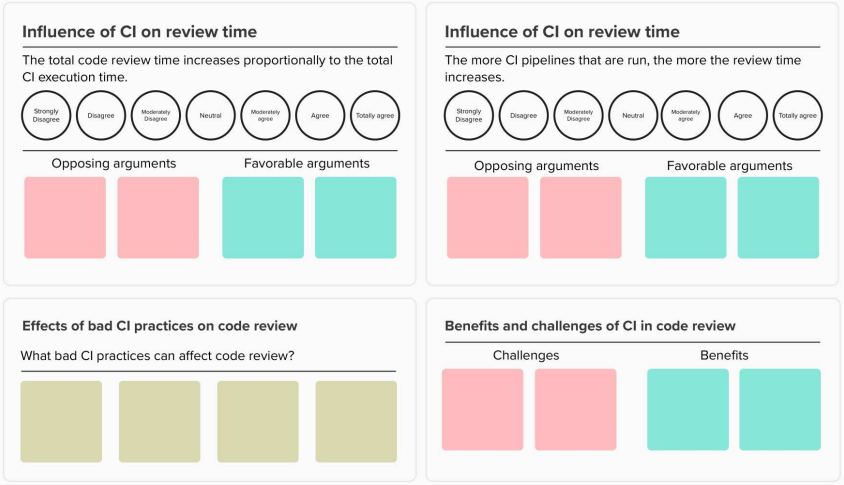
1. **Preparação do ambiente**: o primeiro passo foi escolher as ferramentas para apoiar o grupo focal. Foi utilizada uma ferramenta de quadro branco virtual, Mural [5], para facilitar a visualização dos tópicos em discussão e incentivar a colaboração dos participantes. Dentro da ferramenta, foram criados quadros interativos de acordo com as questões de pesquisa, onde os participantes (1) votaram e expressaram opiniões favoráveis e contrárias a duas declarações com base nos resultados da primeira questão de pesquisa, (2) listaram possíveis efeitos de más práticas IC no processo de revisão de código e (3) listaram benefícios e desafios da IC na RC. A Figura 9 resume os quadros criados dentro da ferramenta. Para facilitar o registro e a transcrição das discussões, foi aplicada a dinâmica do grupo focal via Google Meet [6] e utilizou-se a ferramenta tldv [7] para gravar e transcrever a videoconferência.
2. **Escolha dos participantes**: o segundo passo consistiu na escolha dos participantes do grupo focal. Para isso, foram convidados especialistas que faziam parte da equipe de desenvolvimento dos sistemas analisados para participar voluntariamente de nossa pesquisa do grupo focal. Os convites foram direcionados apenas a participantes que atuaram como revisores de código nos projetos analisados, já que a presente pesquisa se concentra na correlação entre IC e RC. Antes de aplicar o grupo focal, também foi aplicado um questionário para coletar o perfil dos especialistas em termos de (1) nível de educação, (2) experiência em IC e (3) experiência em RC. A Tabela 17 caracteriza os participantes.
3. **Aplicação do grupo focal**: uma vez concluída a fase de planejamento e escolha dos participantes, procedeu-se com o grupo focal. Como mencionado anteriormente, a aplicação ocorreu remotamente via Google Meet, teve duração de cerca de uma hora e foi conduzida da seguinte maneira: (1) primeiramente o tema da pesquisa foi introduzido, bem como a dinâmica do grupo focal e foram respondidas as perguntas dos participantes, (2) então iniciou-se a discussão sobre a influência da IC na RC por meio de duas declarações que

[5] https://www.mural.co/
[6] https://meet.google.com/
[7] https://tldv.io/

os participantes avaliaram, de acordo com sua percepção, em uma escala de “discordo totalmente” a “concordo totalmente”, complementando sua opinião por meio de discussão oral e notas compartilhadas no quadro, (3) em seguida, foi realizada a discussão sobre os efeitos causados por más práticas de IC na RC e, para isso, os participantes tiveram acesso a uma lista de 14 más práticas de IC identificadas anteriormente no ambiente do presente estudo em um estudo anterior (Silva; Bezerra, 2020), aqui os participantes foram incentivados a analisar as más práticas, discutir os possíveis impactos que elas poderiam ter no processo de revisão de código e registrar pontos importantes por meio de notas compartilhadas no quadro, (4) após uma pausa de 5 minutos para um bate-papo informal para relaxar os participantes, foi feito um *brainstorming* para levantar possíveis benefícios e desafios gerados pela IC na RC, para isso, os participantes também fizeram uso das notas compartilhadas no quadro e (5) finalizou-se o grupo focal agradecendo a todos pela disposição em contribuir para a pesquisa, juntamente com um resumo do que foi discutido durante a sessão. A sessão seguiu esse *script* para que os participantes pudessem se concentrar nos tópicos em discussão e evitar questões paralelas que não estivessem diretamente relacionadas à discussão.

4. **Análise dos resultados**: o último passo no grupo focal foi analisar os resultados obtidos a partir das percepções dos participantes. A metodologia para analisar o grupo focal foi baseada na análise qualitativa das notas compartilhadas no quadro complementando a análise com os discursos dos participantes no momento da discussão.

Figura 9 – Mural do grupo focal



Fonte: elaborado pelo Autor.

Tabela 17 – Caracterização dos revisores.

ID	NÍVEL DE EDUCAÇÃO	EXPERIÊNCIA EM CI	EXPERIÊNCIA EM CR
P1	Especialização	Básico	Intermediário
P2	Mestrado	Básico	Básico
P3	Doutorado	Avançado	Avançado
P4	Doutorado	Intermediário	Avançado

Fonte: Elaborado pelo autor.

6.3 Resultados

A seguir os resultados de cada questão de pesquisa são descritos.

6.3.1 Aspectos Relacionados aos Processos de IC e RC (QP_1)

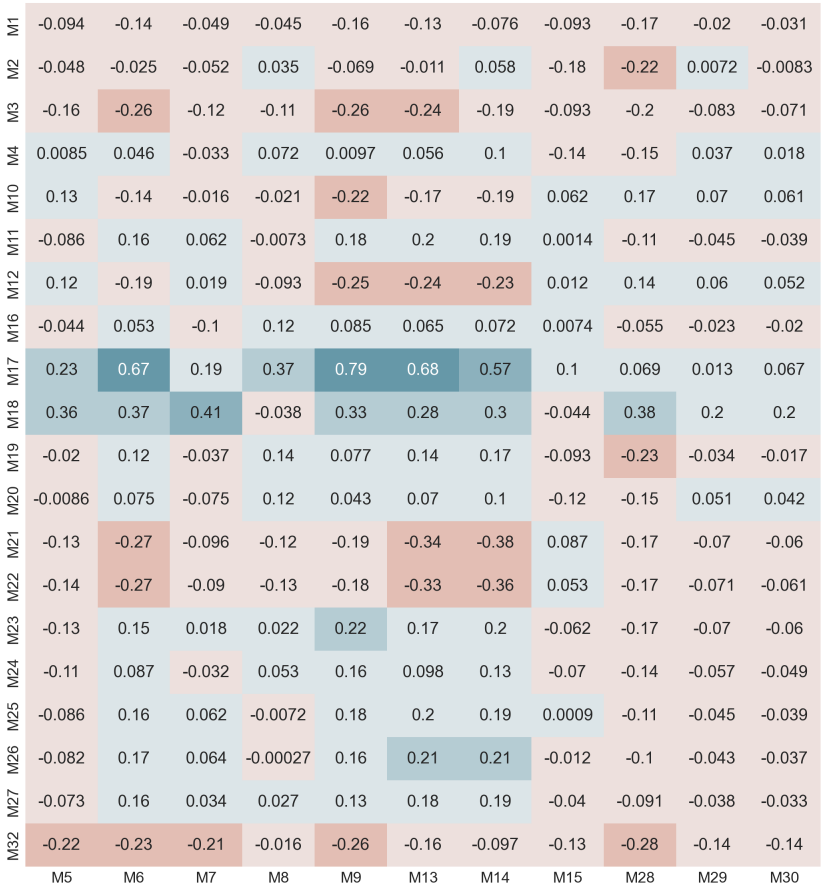
O presente estudo considerou correlações entre métricas de RC e IC de forte a muito forte (ou seja, correlações com um coeficiente maior que 0,6). Assim, um total de 23 correlações foi identificado e se enquadra nessa classificação. A Figura 10 apresenta a matriz de correlação entre as métricas na Tabela 16. Note que algumas métricas não aparecem porque, nesse caso, o algoritmo não pôde estabelecer um valor para o coeficiente de correlação com base no histórico de valores da métrica. Portanto, as linhas e colunas vazias foram omitidas para facilitar a análise dos gráficos [8]. A matriz de correlação está organizada da seguinte forma: o eixo x contém as métricas de IC, e o eixo y compreende as métricas de RC. Quanto maior o valor do coeficiente de correlação, mais escura é a cor associada.

As correlações fortes podem ser visualizadas a partir dos dados compilados nas matrizes de correlação. A Tabela 18 apresenta as correlações fortes entre as matrizes de correlação de projetos individuais e a matriz geral. A tabela também diferencia as correlações diretas (coeficientes positivos) e correlações inversas (coeficientes negativos). Abaixo descreve-se cada correlação encontrada e o que ela pode sugerir no contexto de um ambiente de desenvolvimento de software:

– **Latência da IC e Tempo de revisão**: Essa correlação mede a relação entre a latência ou o tempo de resposta do processo de integração contínua e o tempo total necessário para concluir uma revisão de código. Uma correlação positiva sugere que uma latência de IC mais longa está associada a tempos de revisão mais longos, indicando um possível impacto na eficiência do fluxo de trabalho de desenvolvimento.

[8] O conjunto completo de matrizes de correlação está disponível em https://github.com/ruben-silva-dev/dissertacao-pcomp-ic/tree/main?tab=readme-ov-file#artefatos-do-terceiro-estudo

Figura 10 – Matriz de correlação entre as métricas de IC e RC entre todos os projetos.



Fonte: elaborado pelo Autor.

Tabela 18 – Ocorrências de correlações.

IC	RC	GERAL	22	26	36	39	41	54	55	56	78	133	OCORRÊNCIAS
Latência da IC	Tempo de revisão	0,79	0,93	0,82	0,83			0,87	0,88	0,78		0,63	**8**
	Número de participantes				0,71			0,85	0,71				**3**
	Arquivos adicionados							-0,62					**1**
Builds com falha	Review time							0,79	0,62	0,72			**3**
	Arquivos modificados			0,74			-0,68						**2**
	Número de participantes							0,65					**1**
	Commits auto-fundidos			-0,68									**1**
Jobs com falha	Número de participantes						0,82						**1**
	Review time						0,8						**1**
Builds com sucesso	Arquivos modificados			-0,68									**1**
	Commits auto-fundidos			0,67									**1**
Jobs com sucesso	Review time							0,81	0,7	0,72			**3**
	Arquivos modificados						-0,77						**1**
	Número de participantes							0,84					**1**
	Linhas de código fonte adicionadas						-0,65						**1**
Testes executados	Review time		0,63										**1**
Total de builds	Review time	0,67	0,61	0,67				0,86	0,75	0,7			**6**
	Número de participantes							0,84	0,64				**2**
	Arquivos adicionados							-0,6					**1**
	Arquivos modificados						-0,68						**1**
Total de jobs	Review time	0,68					0,7	0,86	0,76	0,69			**5**
	Número de participantes							0,8	0,61				**2**
	Arquivos adicionados							-0,63					**1**
OCORRÊNCIAS		**3**	**3**	**6**	**2**	**0**	**7**	**13**	**8**	**5**	**0**	**1**	**48**

Fonte: Elaborado pelo autor.

- **Latência da IC e Número de participantes**: Essa correlação examina como o número de participantes envolvidos na revisão de código se relaciona com a latência do processo de integração contínua. Uma correlação negativa indica que uma maior participação está associada a uma latência de IC mais curta, indicando potencialmente que mais revisores contribuem para ciclos mais rápidos de integração e revisão.
- **Latência da IC e Arquivos adicionados**: Essa correlação investiga a relação entre a latência do processo de integração contínua e o número de arquivos adicionados durante o desenvolvimento. Uma correlação positiva sugere que o aumento do número de arquivos adicionados pode resultar em uma latência de IC mais longa, afetando potencialmente a complexidade e a duração das atividades de integração.
- ***Builds* com falha e Tempo de revisão**: Essa correlação explora como a ocorrência de *builds* com falha se relaciona com o tempo total necessário para concluir uma revisão de código. Uma correlação positiva sugere que *builds* com falhas podem levar a tempos de revisão mais longos, indicando um possível atraso causado pela necessidade de resolver problemas de compilação antes de prosseguir com o processo de revisão.
- ***Builds* com falha e Arquivos modificados**: Essa correlação examina como os *builds* com falha estão relacionados ao número de arquivos modificados durante o desenvolvimento. Uma correlação positiva sugere que um número maior de arquivos modificados pode aumentar a probabilidade de falhas nas compilações, o que pode indicar desafios na manutenção da estabilidade do código durante o desenvolvimento.
- ***Builds* com falha e Número de participantes**: Essa correlação investiga como a ocorrência de builds com falha está relacionada ao número de participantes envolvidos na revisão do código. Uma correlação positiva sugere que mais participantes podem resultar em uma identificação e resolução mais eficazes dos problemas de compilação, reduzindo potencialmente a ocorrência de *builds* com falha.
- ***Builds* com falha e *Commits* auto-fundidos**: Essa correlação explora a relação entre a ocorrência de *builds* com falha e o número de *commits* feitos por revisores que, por sua vez, também incorporaram o código. Uma correlação positiva sugere que os *commits* auto-fundidos estão associados a uma maior probabilidade de falhas de compilação, indicando a necessidade de um exame minucioso adicional no processo de revisão.
- ***Jobs* com falha e Número de participantes**: Essa correlação examina como a ocorrência de *jobs* com falha no processo de integração contínua se relaciona com o número de

participantes envolvidos na revisão do código. Uma correlação positiva sugere que um número maior de participantes pode levar a uma identificação e resolução mais eficazes de falhas nos *jobs*.

- ***Jobs* com falha e Tempo de revisão**: Essa correlação investiga a relação entre a ocorrência de *jobs* com falhas e o tempo total necessário para concluir uma revisão de código. Uma correlação positiva sugere que *jobs* com falhas podem resultar em tempos de revisão mais longos, indicando a necessidade de tempo adicional para abordar e retificar problemas relacionados ao *job*.
- ***Builds* com sucesso e Arquivos modificados**: Essa correlação explora como a ocorrência de *builds* bem-sucedidos se relaciona com o número de arquivos modificados durante o desenvolvimento. Uma correlação positiva sugere que mais arquivos modificados podem estar associados a *builds* bem sucedidos, indicando possivelmente uma integração bem-sucedida das alterações de código no sistema.
- ***Builds* com sucesso e *Commits* auto-fundidos**: Essa correlação examina como os *builds* bem-sucedidos estão relacionados ao número de *commits* feitos por revisores que, por sua vez, também incorporaram o código. Uma correlação positiva sugere que os *commits* auto-fundidos estão associados a *builds* bem sucedidos, indicando a capacidade dos revisores de fornecer alterações de código de alta qualidade.
- ***Jobs* com sucesso e Tempo de revisão**: Essa correlação investiga a relação entre a ocorrência de *jobs* bem-sucedidos no processo de integração contínua e o tempo total necessário para concluir uma revisão de código. Uma correlação negativa sugere que os *jobs* bem-sucedidos podem levar a tempos de revisão mais curtos, indicando possivelmente processos de desenvolvimento mais suaves e ciclos de *feedback* mais rápidos.
- ***Jobs* com sucesso e Arquivos modificados**: Essa correlação examina como os *jobs* bem-sucedidos estão relacionados ao número de arquivos modificados durante o desenvolvimento. Uma correlação positiva sugere que mais arquivos modificados podem estar associados a execuções de trabalho bem-sucedidas, indicando a integração bem-sucedida e a estabilidade funcional das alterações.
- ***Jobs* com sucesso e Número de participantes**: Essa correlação explora como a ocorrência de *jobs* bem-sucedidos no processo de integração contínua se relaciona com o número de participantes envolvidos na revisão de código. Uma correlação positiva sugere que mais participantes podem contribuir para execuções de *jobs* bem-sucedidas, indicando

colaboração eficaz e solução de problemas durante o processo de revisão.

- ***Jobs* com sucesso e Linhas de código fonte adicionadas**: Essa correlação investiga a relação entre a ocorrência de *jobs* bem-sucedidos e o número de linhas de código-fonte adicionadas durante o desenvolvimento. Uma correlação positiva sugere que um número maior de linhas adicionadas pode estar associado a execuções de *jobs* bem-sucedidas, indicando a integração e a funcionalidade bem-sucedidas do código adicionado.
- **Testes executados e Tempo de revisão**: Essa correlação mede a relação entre o número de testes executados durante a integração contínua e o tempo total necessário para concluir uma revisão de código. Uma correlação positiva sugere que mais testes podem resultar em tempos de revisão mais longos, indicando a necessidade de mais tempo para validar os resultados dos testes.
- **Total de *builds* e Tempo de revisão**: Essa correlação examina como o número total dos *builds* executados no processo de integração contínua se relaciona com o tempo total necessário para concluir uma revisão de código. Uma correlação positiva sugere que um maior número de *builds* pode estar associado a tempos de revisão mais longos, indicando possivelmente um processo de integração mais complexo.
- **Total de *builds* e Número de participantes**: Essa correlação explora como o número total dos *builds* executados no processo de integração contínua se relaciona com o número de participantes envolvidos na revisão do código. Uma correlação positiva sugere que um número maior de participantes pode contribuir para um número mais significativo de *builds*, indicando, possivelmente, maior colaboração e alterações no código durante o desenvolvimento.
- **Total de *builds* e Arquivos adicionados**: Essa correlação investiga a relação entre o número total dos *builds* executados no processo de integração contínua e o número de arquivos adicionados durante o desenvolvimento. Uma correlação positiva sugere que mais arquivos adicionados podem resultar em mais *builds*, indicando um processo de desenvolvimento mais complexo e iterativo.
- **Total de *builds* e Arquivos modificados**: Essa correlação examina como o número total dos *builds* executados no processo de integração contínua se relaciona com o número de arquivos modificados durante o desenvolvimento. Uma correlação positiva sugere que mais arquivos modificados podem resultar em mais *builds*, indicando a necessidade de integração contínua e validação das alterações de código.

- **Total de *jobs* e Tempo de revisão**: Essa correlação mede a relação entre o número total de *jobs* executados no processo de integração contínua e o tempo total necessário para concluir uma revisão de código. Uma correlação positiva sugere que mais *jobs* podem estar associados a tempos de revisão mais longos, indicando potencialmente uma base de código mais extensa ou um processo de revisão complexo.
- **Total de *jobs* e Número de participantes**: Essa correlação examina como o número total de *jobs* executados no processo de integração contínua se relaciona com o número de participantes envolvidos na revisão do código. Uma correlação positiva sugere que mais participantes podem contribuir para um número mais significativo de *jobs*, indicando maior colaboração e alterações no código durante o desenvolvimento.
- **Total de *jobs* e Arquivos adicionados**: Essa correlação investiga a relação entre o número total de *jobs* executados no processo de integração contínua e o número de arquivos adicionados durante o desenvolvimento. Uma correlação positiva sugere que mais arquivos adicionados podem resultar em mais *jobs*, indicando maior atividade de desenvolvimento e a necessidade de integração contínua.

Ao analisar os dados na Tabela 18, é possível ver que todas as dimensões das métricas de IC e RC na Tabela 16 estão correlacionadas por alguma métrica. Além disso, quando se observa as correlações com o maior número de ocorrências, percebe-se que o tempo de execução da IC (usando a métrica Latência de IC) e a quantidade de trabalho realizado pela IC (usando as métricas Total de *Builds*, Total de *Jobs*, *Builds* com falha e *Jobs* com sucesso) estão fortemente relacionados ao tempo de execução da revisão de código (usando a métrica Tempo de revisão). Essa é uma descoberta interessante, considerando que os processos de IC e RC ocorrem em paralelo e que o revisor espera que os resultados do IC estejam livres de erros antes de iniciar a revisão manual. Outro ponto essencial observado foi sobre as métricas: quando ordenamos o número de ocorrências de cada métrica nas correlações em ordem decrescente, descobriu-se que as métricas de IC e RC mais frequentes foram a latência de IC e o tempo de revisão, respectivamente, o que reforça a teoria de que o tempo de IC pode ter um impacto no tempo de RC.

Descoberta 8 *A carga de trabalho da IC bem como seu tempo de execução podem influenciar o tempo de revisão do código.*

6.3.2 Percepções das equipes de desenvolvimento sobre a correlação entre os processos de IC e RC (QP_2)

Na $\mathbf{QP}_2$, foram analisadas as percepções das equipes de desenvolvimento sobre as correlações encontradas em nossa $\mathbf{QP}_1$. Como resultado, foram obtidas duas afirmações que resumem as correlações: "o tempo total de revisão de código aumenta proporcionalmente ao tempo total de execução da IC"e "quanto mais pipelines de IC são executados, mais o tempo de revisão aumenta". Essas duas afirmações foram apresentadas aos participantes do grupo de foco descrito na Seção 6.2.3.5, onde os participantes usaram suas experiências para dizer o quanto concordavam ou discordavam de cada uma das afirmações e para expressar argumentos a favor e contra cada uma delas. A seguir, as afirmações são analisadas separadamente.

6.3.2.1 O tempo total de revisão do código aumenta proporcionalmente ao tempo total de execução da IC

Na votação da primeira afirmação, foi obtido um resultado mais inclinado à discordância, com dois votos para a opção “discordo moderadamente” e um para as opções “neutro” e “concordo moderadamente”, respectivamente. Considerando apenas os dados, pode-se dizer que há uma contradição nos resultados da $\mathbf{QP}_1$, pois a matriz de correlação indica uma correlação entre IC e RC, especificamente sobre o tempo de execução de ambos os processos. Entretanto, quando são considerados também os comentários feitos pelos participantes do grupo focal, pode se encontrar uma possível explicação para a correlação entre IC e RC.

Entre os comentários feitos, pode-se destacar alguns, como: *“posso identificar situações em que a revisão de código seria relativamente simples, mesmo que o pipeline seja demorado; no entanto, concordo que, se o processo de revisão depender de etapas na IC, então sim”*, outro comentário foi *“em alguns casos, o revisor pode precisar esperar que o processo de compilação termine e que os relatórios sejam gerados na IC antes de iniciar a revisão”*. Esses comentários sugerem que, se o processo de RC depender muito do processo de IC no contexto da organização, ou seja, se o revisor sempre tiver que esperar que o resultado do IC esteja livre de erros antes de iniciar a revisão, os tempos de execução de ambos os processos poderão estar correlacionados.

6.3.2.2 Quanto mais pipelines de IC forem executados, mais o tempo de revisão aumentará

Os resultados obtidos para essa declaração foram semelhantes aos da declaração anterior. Na votação, foi obtida a mesma pontuação da última declaração, com dois votos para "discordo moderadamente" e um voto para "neutro" e "concordo moderadamente", respectivamente. Na sessão de comentários, foram obtidas percepções mais relevantes que corroboram as descobertas na declaração anterior.

Aqui estão alguns comentários feitos pelos participantes que se mostraram relevantes para enriquecer a análise: *"o processo pode resultar na revisão apenas dos problemas marcados como concluídos pelo desenvolvedor, independentemente dos pipelines envolvidos"* e *"quando o processo de pipeline atrasa, surge a incerteza sobre a viabilidade de revisar o código antecipadamente, pois o código pode não passar nos testes de pipeline"*. Esses dois comentários mostram dois aspectos diferentes da correlação entre IC e RC: No primeiro comentário, nota-se um cenário em que o revisor não precisa esperar pelo resultado da IC e já realiza sua análise com base no que o desenvolvedor sinaliza como pronto, é o caso em que os processos de IC e RC são executados completamente em paralelo e o desenvolvedor lida com os problemas relatados pela IC e RC ao mesmo tempo; no segundo comentário, ocorre um cenário em que o revisor prefere esperar que o resultado da IC esteja livre de erros antes de prosseguir com a análise e, neste caso, o desenvolvedor terá que lidar primeiro com os problemas relatados pela IC e depois com os da RC.

O objetivo do presente trabalho não é avaliar qual das duas abordagens de uso de IC e RC é mais adequada (paralela ou sequencial), mas sim analisar a relação entre os dois processos. No entanto, foi possível ver que, de acordo com a visão dos participantes do grupo de foco, a abordagem sequencial parece gerar uma correlação mais significativa entre o tempo de IC e RC. Portanto, é razoável dizer que a IC e a RC estão correlacionadas se o modelo de trabalho adotado pela organização priorizar a solução dos problemas levantados pela IC antes de iniciar o processo de RC.

Descoberta 9 *Os tempos de execução da IC e da RC são mais correlacionadas quando os dois processos ocorrem sequencialmente.*

6.3.3 Acerca dos efeitos causados pelas más práticas de IC no processo de RC (QP_3)

Nesta seção, os comentários feitos pelos participantes do grupo focal são analisados dividindo a análise de acordo com as más práticas citadas por eles. Ao todo, os participantes comentaram sobre dez das quatorze más práticas consideradas neste estudo. A Tabela 19 apresenta as más práticas consideradas, organizadas em ordem decrescente pela quantidade de comentários feitos pelos revisores durante o grupo focal. A primeira coluna apresenta um identificador que é utilizado para se referir às más práticas durante a discussão, a segunda coluna descreve as más práticas, e a última coluna informa a quantidade de comentários associados a cada má prática.

Tabela 19 – Más práticas analisadas.

ID	MÁ PRÁTICA	COMENTÁRIOS
MP1	Branches divergentes	3
MP2	Testes ausentes em *feature branches*	2
MP3	Algumas tarefas de *pipelines* são iniciadas manualmente	2
MP4	*Scripts* de *build* longos	2
MP5	Os desenvolvedores não têm controle total do ambiente	1
MP6	Notificações de falhas são enviadas apenas para equipes/desenvolvedores alocados na tarefa	1
MP7	Falta de teste em um ambiente de produção	1
MP8	O hardware do servidor IC é usado para finalidades diferentes, além da execução da estrutura de IC	1
MP9	Ferramentas externas são usadas com suas configurações padrão	1
MP10	*Quality gates* são definidos sem os desenvolvedores, considerando apenas o que é ditado pelo cliente	1
MP11	*Feature branches* são usadas em vez de *feature toggles*	0
MP12	Os casos de teste não são organizados em pastas com base em seus propósitos	0
MP13	Um *build* falha devido a alguma instabilidade na execução que não deveria ocorrer	0
MP14	Desenvolvedores e operadores são mantidos como funções separadas	0

Fonte: Elaborado pelo autor.

Analisando a Tabela 19, é perceptível que algumas das más práticas chamaram mais a atenção dos revisores. As quatro primeiras más práticas foram as que mais se destacaram, recebendo mais de um comentário. Dentre estas, a MP1, referente às "*branches* divergentes", foi mencionada em três comentários. As más práticas de 5 a 10 receberam apenas um comentário, enquanto as quatro restantes não foram comentadas.

É interessante notar que as quatro primeiras más práticas receberam mais atenção e comentários dos revisores. Isso pode indicar que essas más práticas são percebidas como mais impactantes ou mais frequentes no contexto da integração contínua e podem estar mais evidentes no dia a dia dos revisores ou da equipe como um todo. Existem algumas possíveis razões para essa maior atenção:

1. **Impacto Direto e Evidente**: As más práticas relacionadas a divergência de *branches*, falta de testes em *branches* de novas funcionalidades, passos manuais na pipeline e

scripts longos podem ter um impacto mais direto e imediato no fluxo de trabalho dos desenvolvedores e revisores. Isso pode resultar em problemas evidentes que afetam diretamente a revisão de código.

2. **Frequência de Ocorrência**: Essas más práticas podem ser mais comuns ou recorrentes no ambiente de desenvolvimento específico em que os revisores trabalham. A frequência de problemas relacionados a essas práticas pode levá-los a expressar suas preocupações com mais detalhes.
3. **Consequências Visíveis**: As más práticas que geram conflitos, atrasos ou dificuldades operacionais podem ser percebidas mais claramente pelos revisores, o que leva a comentários mais detalhados sobre seus efeitos negativos.
4. **Conhecimento e Experiência**: Os revisores podem ter mais experiência ou conhecimento específico sobre essas primeiras más práticas, permitindo que expressem melhor suas preocupações e sugiram possíveis soluções.

Ao analisar os comentários dos revisores durante a sessão do grupo focal, percebe-se que as más práticas de IC podem gerar alguns transtornos que podem impactar diretamente ou indiretamente na RC. A seguir, as más práticas que obtiveram os comentários mais relevantes são analisadas[9].

- **MP1 - *Branches* divergentes**: A divergência de *branches* pode levar a conflitos que exigem resolução manual, o que pode ser prejudicial durante a revisão de código. Os revisores relataram que a MP1 ocorre frequentemente: *"branches divergentes acontecem muito com a demora dos desenvolvedores em atualizar suas branches com o padrão."* Outro problema apontado foi a falta de experiência por parte dos desenvolvedores para resolver adequadamente os conflitos antes de enviar o código para revisão, momento em que a *branch* deveria estar livre de conflitos: *"branches divergentes são especialmente complicados quando os merges acontecem e a equipe não possui experiência para resolvê-los adequadamente. Idealmente, os merges não deveriam gerar conflitos."* Portanto, é possível afirmar que a MP1 pode atrasar o processo de RC, como mencionado por um dos revisores: *"Conflitos de código demandam tempo para resolução."*
- **MP2 - Testes ausentes em *feature branches***: A ausência de testes adequados pode aumentar a carga de trabalho da equipe de revisão, que precisa assegurar que as novas funcionalidades não quebrem o que já estava funcionando. Os revisores reconhecem que

[9] A lista completa dos comentários pode ser encontrada através do link: https://github.com/ruben-silva-dev/dissertacao-pcomp-ic/blob/main/estudo-3/focus-group.pdf

"uma pipeline sem testes é uma pipeline incompleta.", o que pode prejudicar o processo de RC, forçando os revisores a desempenhar um nível mais rigoroso de revisão para detectar erros e garantir a estabilidade do software, resultando em um aumento no tempo de revisão. É importante lembrar que as mudanças feitas pelos desenvolvedores podem afetar diversas áreas, como destacado por um dos revisores: *"a falta de testes de regressão afeta a garantia da qualidade do software, já que a integração de novas funcionalidades pode impactar no que já estava funcionando anteriormente."* Neste último caso, a carga de trabalho dos revisores aumentaria ainda mais, pois precisariam examinar detalhadamente o código para garantir que as funcionalidades anteriores não foram prejudicadas pelas mudanças.

- **MP3 - Algumas tarefas de pipelines são iniciadas manualmente**: Os passos manuais podem causar atrasos no processo de entrega e impactar o cronograma de revisões de código. Esses atrasos podem ser atribuídos à necessidade de intervenção humana dentro do processo de IC, como afirma um dos revisores: *"passos manuais no pipeline podem levar a atrasos, já que dependem de alguém (humano) para executar algo."* Outro ponto desfavorável aos passos manuais é o fato de que o processo de IC perde a continuidade, uma vez que o *pipeline*, em algum momento, aguarda uma ação manual, conforme mencionado em um dos comentários: *"passos manuais são feitos para serem esquecidos. Além disso, a integração fica parcialmente contínua."*
- **MP4 - *Scripts* de *build* longos**: Esta má prática pode resultar em atrasos no ciclo de desenvolvimento, incluindo na fase de revisão de código, devido a *scripts* demorados. Uma das razões para essa lentidão pode estar relacionada à falta de otimização dos *scripts*, como uma estrutura de cache deficiente ou inexistente. Outro fator pode ser a alta complexidade do próprio *script*, que, em alguns casos, pode ser reduzida. Em qualquer caso, é evidente que *"scripts longos ou com tarefas demoradas podem ser lentos e atrasar o processo."* Isso pode até mesmo *"impedir a continuidade do trabalho"*, tornando *"essa má prática ainda mais grave"*. Neste sentido, é plausível afirmar que a MP4 pode aumentar o tempo necessário para iniciar o processo de RC.
- **MP5 - Os desenvolvedores não têm controle total do ambiente**: Esta má prática pode influenciar indiretamente a revisão de código, uma vez que os desenvolvedores podem enfrentar desafios na replicação de ambientes específicos para testes e revisão. Um comentário oposto a essa má prática foi feito: *"Discordo. Acredito que o desenvolvedor*

precisa, de fato, ter certo controle do ambiente, mas não necessariamente o controle total do mesmo." Isso pode estar relacionado ao ambiente do estudo, onde os desenvolvedores atuam como estagiários e, portanto, não têm acesso aos recursos avançados da organização. Além disso, eles podem não possuir um conhecimento abrangente das tecnologias adotadas nos projetos. No entanto, é importante ressaltar que ter controle do ambiente pode aumentar a carga de trabalho dos revisores, pois eles terão que analisar as mudanças sem considerar fatores específicos relacionados ao ambiente do sistema.

– **MP6 - Notificações de falhas são enviadas apenas para equipes/desenvolvedores alocados na tarefa**: A visibilidade limitada dos problemas identificados pela IC pode afetar o processo de RC, uma vez que as notificações de falhas são direcionadas apenas para os desenvolvedores ou equipes que optaram por recebê-las. Outras partes interessadas podem não estar cientes de problemas críticos no processo de IC, incluindo revisores que não estão diretamente envolvidos na configuração das notificações. Um problema detectado durante a IC pode resultar em uma falha posterior no código em revisão, demandando mais tempo e esforço dos revisores para identificar e corrigir. A correção desses problemas pode sofrer atrasos, já que os membros não notificados podem não tomar medidas imediatas para corrigir ou investigar as questões levantadas. Um dos revisores concordou ao afirmar que *"enviar notificações apenas para os desenvolvedores que optaram por recebê-las não é o ideal"*, sugerindo ainda que *"essas notificações possam ser direcionadas para partes interessadas específicas, como uma equipe específica ou um grupo de desenvolvedores"*. Em todo caso, o ideal é que, pelo menos, a equipe responsável pelo projeto seja notificada das falhas.
– **MP7 - Falta de teste em um ambiente de produção**: A ausência de um ambiente de testes similar ao de produção pode resultar em problemas que só são identificados em estágios posteriores, incluindo durante a revisão de código. Como um dos revisores mencionou, esta é *"outra má prática clássica. Um ambiente de testes similar ao de produção evita que alguns problemas ocorram somente em produção"*. Sendo assim, é plausível afirmar que a MP7 pode prejudicar a RC, transferindo para os revisores a responsabilidade de identificar os erros não reportados pela IC, o que também não impede que os problemas cheguem ao ambiente de produção.
– **MP9 - Ferramentas externas são usadas com suas configurações padrão**: O uso de configurações padrão pode não atender completamente às demandas do projeto, resultando

em comportamentos inesperados ou inadequados durante o processo de revisão de código. Isso pode dificultar a compreensão do código pelos revisores ou introduzir problemas que podem passar despercebidos durante a revisão. Um dos revisores comentou que *"é uma falha comum manter as configurações padrões que nem sempre atendem as demandas do time."* Isso é verdade, pois as configurações padrão podem não ser otimizadas para o contexto específico do projeto, aumentando os riscos de erros, incompatibilidades ou falhas durante a integração das alterações revisadas. Isso pode afetar negativamente a confiabilidade e a estabilidade do código. Portanto, é crucial personalizar as configurações dessas ferramentas externas para atender às necessidades específicas do projeto. Isso pode melhorar a qualidade do código revisado, reduzir ambiguidades e simplificar a revisão para os membros da equipe.

Através dos comentários dos revisores, foi possível perceber que as más práticas de IC podem impactar significativamente o processo de revisão, resultando em possíveis atrasos, aumento da carga de trabalho da equipe de revisão e riscos de problemas não detectados. Neste sentido, surge a necessidade de lidar com essas más práticas e mitigar seus efeitos negativos no processo de RC.

Descoberta 10 *As más práticas em IC impactam o processo de RC com atrasos, aumento da carga de trabalho e maior chance de problemas não identificados.*

6.3.4 Benefícios e desafios gerados pela IC na RC (QP_4)

Como foi explicado na Seção 6.2.3.5, a última etapa do grupo focal consistiu em um *brainstorming* no qual os participantes tiveram a oportunidade de comentar sobre os benefícios e desafios por eles percebidos do processo de IC sobre o processo de RC. A seguir são discutidos separadamente os benefícios e desafios levantados pelos revisores.

6.3.4.1 Benefícios

Durante a discussão, os participantes destacaram aspectos da IC que contribuem diretamente para aprimorar o processo de RC, apoiados por comentários dos revisores:

– **Identificação Proativa de Problemas:** Ferramentas de sinalização na IC permitem uma

identificação mais rápida de áreas problemáticas no código. Isso oferece aos revisores um foco direcionado durante a RC, otimizando o tempo gasto na análise de trechos críticos.

> *"Ferramentas de sinalização de problemas facilitam a revisão."*

– **Detecção Antecipada de *Bugs*:** Com testes automatizados e frequentes na IC, os *bugs* são identificados precocemente, o que reduz a quantidade de problemas que podem passar despercebidos durante a RC, contribuindo para a qualidade do código.

> *"Ajuda a encontrar bugs de maneira mais rápida, visto que os testes se tornam mais frequentes e automatizados."*

– **Distribuição Eficiente de Atualizações:** A automatização da implantação de novas versões na IC permite uma distribuição ágil e controlada das atualizações para os clientes. Isso não só mantém o software atualizado, mas também minimiza erros devido à implantação manual.

> *"Facilita na distribuição de novas atualizações do produto para o cliente."*

– **Antecipação de Problemas na *Pipeline*:** Identificar problemas na pipeline da IC antes da RC auxilia na priorização de correções durante a revisão, aumentando a eficiência do processo ao direcionar a atenção para áreas críticas.

> *"Alguns problemas podem ser antecipados em passos da pipeline, ajudando na priorização de tarefas e no gerenciamento de tempo."*

– **Segurança Automatizada:** Verificações automáticas de segurança e vulnerabilidade integradas à IC evitam a introdução de falhas críticas no código, reduzindo significativamente a necessidade de correções durante a RC.

> *"Passos que envolvem a checagem de segurança e vulnerabilidade são essenciais e podem ser feitos de forma automática. Dificilmente tais problemas seriam encontrados na revisão."*

– **Redução de Tarefas Repetitivas:** A automação na IC minimiza tarefas manuais repetitivas, permitindo que os desenvolvedores se concentrem mais na análise qualitativa durante a RC, em vez de lidar com procedimentos monótonos.

> *"Uma boa IC simplifica a realização de tarefas repetitivas por meio de automação."*

Os benefícios da IC na RC são pilares essenciais para fortalecer a eficácia e qualidade do ciclo de desenvolvimento de software. Ao antecipar problemas, facilitar a distribuição de atualizações e reduzir tarefas manuais, a IC não apenas aprimora a revisão de código, mas também estabelece uma base sólida para um processo de desenvolvimento ágil e eficiente. Essa simbiose entre a IC e a RC promove não apenas a detecção proativa de erros, mas também a melhoria contínua do produto final.

Descoberta 11 *A IC agrega valor à RC ao antecipar problemas, facilitar a distribuição de atualizações e reduzir tarefas manuais, fortalecendo a eficiência do ciclo de desenvolvimento de software.*

6.3.4.2 Desafios

Os desafios apontados pelos participantes em relação à IC e sua influência na RC foram corroborados pelos comentários dos revisores:

- **Garantia da Qualidade do Código:** Mesmo com a IC, garantir a qualidade do código durante a RC permanece desafiador devido à complexidade dos problemas que podem não ser totalmente detectados pelos testes automatizados.

  "Garantir a qualidade do código é desafiador independentemente do pipeline. Uma pipeline bem sucedida ajuda, mas a revisão por si só é um desafio."

- **Resolução de Conflitos:** Conflitos no código, especialmente ao integrar diferentes *branches*, podem causar atrasos na RC, demandando mais tempo para a resolução de problemas durante essa etapa.

  "Conflitos no código podem atrasar bastante o processo."

- **Implementação de Testes Complexos:** Implementar testes na IC, principalmente aqueles relacionados a bancos de dados e cenários mais complexos, pode ser desafiador e requer um esforço adicional para garantir uma cobertura eficaz durante a RC.

  "Implementar diferentes tipos de testes, especialmente quando lidam com bancos de dados na IC é bem desafiador."

- **Adequação do Processo:** Alinhar o processo de IC com o de RC é essencial, e a falta de sincronização pode criar lacunas no fluxo de trabalho, afetando a eficiência da revisão.

"O processo usado afeta IC e revisão. Adequação de processo é desafiador."

– **Riscos na Adoção do Processo:** A integração de um novo processo de IC pode trazer riscos, exigindo uma transição cuidadosa e uma adaptação adequada para evitar interrupções na RC.

"A adequação a qualquer processo é sempre um risco."

Os desafios enfrentados ao integrar a IC à RC são um reflexo da complexidade inerente ao aprimoramento contínuo do processo de desenvolvimento. Superar obstáculos, como a garantia da qualidade do código, a resolução de conflitos e a harmonização de processos, requer uma abordagem holística e adaptativa. A compreensão desses desafios não apenas amplia a percepção sobre as nuances da integração da IC à RC, mas também destaca a importância de soluções estratégicas e flexíveis para o sucesso do desenvolvimento de software.

Descoberta 12 *A utilização conjunta da IC e RC apresenta desafios, como garantir a qualidade do código, resolver conflitos e harmonizar processos. Isso demanda abordagens estratégicas e adaptativas para otimizar o ciclo de desenvolvimento de software.*

6.4 Discussão

Os resultados obtidos na análise da QP_1 revelaram uma correlação significativa entre o tempo de execução da IC e a duração da RC. A descoberta dessa correlação aponta para uma interdependência entre esses processos no ciclo de desenvolvimento de software. A correlação foi evidenciada tanto quantitativamente, por meio da análise estatística dos dados, quanto qualitativamente, pelas percepções compartilhadas pelas equipes de desenvolvimento durante o grupo focal.

Essa relação de dependência entre IC e RC sugere que um tempo prolongado de execução na IC está associado a uma extensão do período dedicado à RC. Quando ambos os processos são realizados sequencialmente, os resultados indicam uma correlação mais forte, destacando que a integração eficiente entre IC e RC é crucial para a obtenção de bons resultados no ciclo de desenvolvimento.

As percepções compartilhadas pelas equipes de desenvolvimento reforçaram os resultados encontrados na QP_1. A correlação entre os tempos de execução da IC e da RC foi

confirmada através das experiências relatadas pelos participantes do grupo focal. Ficou evidente que essa correlação tende a ser mais proeminente quando os processos de IC e RC são realizados sequencialmente, enfatizando a importância melhorar esses dois processos para garantir a sua eficiência no ciclo de desenvolvimento.

Estas descobertas reforçam a ideia de que uma estratégia de IC bem integrada à RC pode ser crucial para otimizar o processo de desenvolvimento de software, influenciando diretamente na redução de tempo e na melhoria da qualidade do código.

Os resultados da QP_3 destacaram os impactos negativos das más práticas de IC no processo de RC. As deficiências na realização de testes adequados, integração ineficiente de código e inconsistências nos processos de IC foram identificadas como fatores determinantes que prejudicam a qualidade e eficiência da RC. Essas descobertas ressaltam a importância de práticas sólidas de IC para a otimização do processo de RC, minimizando retrabalhos e aprimorando a qualidade do código revisado.

A identificação desses efeitos negativos proporciona um ponto de partida para melhorias no ciclo de desenvolvimento, ressaltando a necessidade de implementar práticas de IC mais robustas para garantir um impacto positivo na RC.

A análise da QP_4 identificou uma série de benefícios associados à integração da IC na RC, como a identificação proativa de problemas, detecção antecipada de *bugs* e distribuição eficiente de atualizações. Esses benefícios estão alinhados com as práticas recomendadas de desenvolvimento de software e contribuem para aprimorar a qualidade do código e acelerar o processo de revisão.

No entanto, os desafios enfrentados, como a garantia da qualidade do código, resolução de conflitos e adequação dos processos, exigem abordagens estratégicas para serem superados. A compreensão desses benefícios e desafios é fundamental para uma implementação eficaz da IC no contexto da RC, visando a melhoria contínua do ciclo de desenvolvimento.

Em resumo, os resultados obtidos em cada questão de pesquisa ressaltam a interdependência entre IC e RC, sublinhando a necessidade de uma integração eficiente e práticas sólidas de IC para otimizar o desenvolvimento de software.

6.5 Ameaças à validade

Discutimos as ameaças à validade do estudo de acordo com o modelo proposto por Wohlin *et al.* (2012) a seguir.

6.5.0.1 Validade Interna

1. Fatores de confusão não identificados: Existem possíveis variáveis não consideradas que podem influenciar a relação entre os processos de IC e RC, afetando os resultados. Para tratar esta ameaça, foram tomados como base os trabalhos existentes na literatura buscando entender ao máximo o ambiente do estudo para selecionar o conjunto de métricas abrangente o suficiente para capturar as diversas nuances da relação entre IC e RC.
2. Viés de seleção: A seleção dos participantes para o grupo focal pode introduzir viés, pois as opiniões expressas podem não representar totalmente a diversidade de experiências e perspectivas existentes nas equipes de desenvolvimento. Entretanto, considerando o contexto do estudo no qual o foco se concentra nos processos de IC e RC e o ambiente de nossos parceiros na indústria, onde a equipe de revisão se limita aos líderes técnicos dos projetos, foi considerado que o perfil dos participantes do grupo focal corresponde ao esperado no contexto do trabalho.
3. Instrumentos de coleta de dados: A interpretação dos resultados pode ser influenciada pela subjetividade na análise das respostas dos participantes e nas informações extraídas das ferramentas de monitoramento dos processos. Para mitigar esta ameaça, foi realizada a validação do *script* de coleta de dados por um especialista e, no caso da aplicação e análise dos dados do grupo focal, foi tomado como base o trabalho de (Uchôa *et al.*, 2020) que obteve bons resultados na sua aplicação.

6.5.0.2 Validade Externa

1. Generalização dos resultados: Os resultados obtidos podem não ser generalizáveis para todas as organizações de desenvolvimento de software, pois os contextos e práticas podem variar significativamente. Sendo assim, pode-se considerar a validade dos resultados apenas para organizações com ambiente similar ao aqui abordado, sendo caracterizado pela existência de más práticas de IC, equipe de desenvolvimento de nível não avançado e equipe de revisão com nível mais madura.
2. Tempo de coleta de dados: Mudanças nos processos de IC e RC ao longo do tempo podem afetar a representatividade dos resultados, especialmente se houver alterações substanciais após a coleta de dados. Neste caso, de forma similar à questão da generalização, pode-se considerar a validade somente para um ambiente similar ao que foi analisado no presente

trabalho.

6.5.0.3 Validade de Construção

1. Definições e métricas: As definições de IC, RC e suas métricas utilizadas podem diferir de outras interpretações, o que pode impactar a comparabilidade dos resultados com outros estudos. Para tratar esta ameaça foram tomados como base os estudos existentes, sobretudo o trabalho de Chen *et al.* (2019), para a seleção das métricas de IC e RC a serem utilizadas. Fpo feita ainda a suplementação do conjunto de métricas com algumas métricas relevantes no contexto do trabalho e que não foram abordadas por Chen *et al.* (2019).
2. Amostragem de dados: A seleção dos dados para análise pode não capturar todos os aspectos dos processos de IC e RC, o que pode limitar a compreensão abrangente desses processos. Neste caso, buscando obter a maior amostragem possível, optou-se por coletar os valores das métricas desde o início dos projetos.

6.5.0.4 Validade de Conclusão

1. Interpretação dos resultados: A inferência das relações causais entre IC e RC pode ser limitada, pois os resultados são baseados em correlações identificadas e percepções dos participantes. Para esta ameaça, foi considerado que, dentro do contexto de projetos industriais, a análise quantitativa/qualitativa foi abrangente o suficiente para proporcionar validade às conclusões.
2. Limitações do escopo: O estudo pode não abranger todas as nuances e complexidades dos processos de IC e RC, reduzindo a amplitude das conclusões alcançadas. Para tratar esta ameaça, foi considerado que, em virtude da utilização de uma metodologia baseada nos resultados de trabalhos já existentes na literatura e considerando as características dos projetos aqui analisados, os resultados são válidos. Entretanto, há de se levar em conta que um trabalho similar no contexto de projetos *open-source* é importante para verificar os resultados.

6.6 Conclusão

O estudo apresentado teve como objetivo investigar a relação intrínseca entre os processos de IC e RC em ambientes de desenvolvimento de software no contexto de sistemas da

industria. A análise abrangente das quatro questões de pesquisa ($\mathbf{QP}_1$ a $\mathbf{QP}_4$) proporcionou uma compreensão mais profunda sobre como esses processos se interconectam, os impactos dessa relação e os desafios enfrentados pelas equipes de desenvolvimento.

A $\mathbf{QP}_1$ explorou a correlação entre o tempo de execução da IC e o tempo total de revisão de código, revelando uma associação significativa entre esses dois processos. A descoberta principal indica que o tempo de execução da IC influencia diretamente a duração da RC, sugerindo que a eficiência e a agilidade na IC podem impactar positivamente a revisão de código.

Ao investigar as percepções das equipes de desenvolvimento sobre essa correlação na $\mathbf{QP}_2$, percebeu-se que essa interação é mais notável quando os tempos de IC e RC ocorrem de forma sequencial. Essa observação destaca a importância de uma integração harmoniosa entre os processos para otimizar a eficácia e minimizar possíveis gargalos no ciclo de desenvolvimento.

No contexto das más práticas de IC e seus impactos na RC ($\mathbf{QP}_3$), descobriu-se que más práticas de IC podem resultar em efeitos adversos na revisão de código. Isso pode incluir desde a introdução de *bugs* não detectados durante a IC até a sobrecarga de trabalho para os revisores, afetando diretamente a qualidade do código revisado.

Aprofundando a análise dos benefícios e desafios associados à IC na RC ($\mathbf{QP}_4$), observou-se que a IC oferece vantagens substanciais, como identificação proativa de problemas, detecção antecipada de *bugs* e distribuição eficiente de atualizações. No entanto, ela enfrenta desafios consideráveis, como a garantia da qualidade do código, resolução de conflitos e adequação de processos, exigindo estratégias adaptativas para mitigar tais obstáculos.

É crucial ressaltar que essa interdependência entre IC e RC é complexa e abrange aspectos técnicos e humanos. A implementação cuidadosa e a integração adequada desses processos são cruciais para colher os benefícios potenciais e superar os desafios identificados neste estudo.

Em resumo, este estudo forneceu uma visão abrangente e *insights* valiosos sobre a relação entre IC e RC no desenvolvimento de software. É importante destacar a importância de considerar esses processos de forma integrada e estratégica, visando aprimorar continuamente a qualidade do código, a eficiência do ciclo de desenvolvimento e, por conseguinte, a satisfação do cliente.

7 OPEN SCIENCE

Este trabalho investigou os efeitos das más práticas de IC em projetos industriais de desenvolvimento de *software*. Identificar, compreender e mitigar esses impactos não apenas é crucial para a eficiência e qualidade dos produtos finais, mas também se alinha aos princípios fundamentais da *Open Science* (Ciência Aberta).

A *Open Science* preconiza a transparência, a colaboração e o acesso aberto aos resultados da pesquisa. Ao reconhecer a importância de divulgar não apenas os resultados, mas também os métodos e dados subjacentes, este estudo busca contribuir para a promoção desses valores na comunidade científica e tecnológica.

Ao abordar questões como o uso inadequado de *feature branches*, *branches* divergentes e a falta de controle do ambiente pelos desenvolvedores, este trabalho não apenas analisa os desafios enfrentados na prática da IC, mas também busca fornecer *insights* que possam ser amplamente compartilhados e debatidos pela comunidade.

Além disso, ao considerar o impacto da IC na qualidade do *software* e sua interação com a RC, este estudo destaca a importância de uma abordagem transparente e colaborativa para aprimorar os processos de desenvolvimento de *software*.

Dessa forma, este trabalho não apenas contribui para o avanço do conhecimento em sua área específica, mas também promove os ideais da *Open Science*, ao buscar a transparência, replicabilidade e acessibilidade dos resultados e métodos de pesquisa. Neste capítulo, são discutidas as estratégias adotadas para integrar esses princípios à metodologia e prática deste estudo.

7.1 Introdução à Open Science

A *Open Science*, ou Ciência Aberta, é uma abordagem na qual os princípios de transparência, colaboração e acessibilidade são aplicados ao processo de pesquisa científica. Em contraste com o modelo tradicional de pesquisa, no qual os resultados são frequentemente compartimentados e acessíveis apenas a um pequeno número de especialistas, a *Open Science* busca tornar a pesquisa e seus resultados amplamente disponíveis para a comunidade científica e o público em geral.

Um dos pilares da *Open Science* é o compartilhamento de dados. Isso envolve disponibilizar os conjuntos de dados subjacentes a um estudo para que outros pesquisadores

possam validar os resultados, realizar análises adicionais e construir sobre o trabalho existente. Com o compartilhamento de dados, a transparência e a confiabilidade da pesquisa são aumentadas, permitindo uma maior verificação e replicabilidade dos resultados.

Além do compartilhamento de dados, a *Open Science* também defende o uso de código aberto. Isso significa disponibilizar o código de software utilizado na pesquisa de forma aberta, permitindo que outros pesquisadores examinem, modifiquem e reutilizem o código para seus próprios fins. O código aberto não apenas facilita a reprodução dos resultados, mas também promove a colaboração e o desenvolvimento conjunto de ferramentas e métodos de pesquisa.

Outro aspecto importante da *Open Science* é o acesso aberto a publicações científicas. Tradicionalmente, muitos artigos científicos são publicados em revistas de acesso restrito, tornando difícil ou caro para pesquisadores e o público em geral acessarem o conhecimento científico mais recente. O acesso aberto, por outro lado, permite que os artigos sejam disponibilizados gratuitamente para leitura e download, promovendo um maior compartilhamento e disseminação do conhecimento.

Por fim, a *Open Science* promove a colaboração aberta entre pesquisadores, incentivando a troca de ideias, dados e recursos entre diferentes instituições e disciplinas. Ao facilitar a colaboração, a *Open Science* permite que pesquisadores trabalhem juntos para resolver problemas complexos e fazer avanços significativos em suas áreas de estudo.

Em resumo, a *Open Science* é importante porque promove a transparência, replicabilidade e acessibilidade na pesquisa científica, permitindo uma maior confiabilidade dos resultados, uma disseminação mais ampla do conhecimento e uma colaboração mais eficaz entre os pesquisadores. Ao adotar os princípios da *Open Science*, os pesquisadores podem contribuir para uma comunidade científica mais aberta, inclusiva e produtiva.

7.2 Contextualização

A pesquisa em ciência da computação e engenharia de software enfrenta desafios únicos em relação à transparência, replicabilidade e confiabilidade dos resultados. No contexto dos projetos de software industriais, onde a IC desempenha um papel crucial na entrega eficiente e na qualidade do produto final, esses desafios são especialmente significativos.

A demanda por transparência na pesquisa científica nunca foi tão premente. À medida que a complexidade dos projetos de software aumenta, torna-se essencial entender e mitigar os riscos associados às más práticas de IC, que podem impactar diretamente a eficiência

operacional, a qualidade do software e, consequentemente, a satisfação do cliente.

O modelo tradicional de publicação científica, com suas barreiras de acesso e revisão limitada, muitas vezes não atende às necessidades da comunidade de engenharia de software. A falta de transparência nos métodos de pesquisa e a dificuldade em acessar dados e resultados relevantes podem levar a uma lacuna significativa entre a teoria e a prática, dificultando a adoção de melhores práticas e a inovação na indústria de software.

Nesse cenário, a *Open Science* surge como uma resposta fundamental aos desafios enfrentados pela pesquisa em engenharia de software. Ao promover os princípios de transparência, colaboração e acessibilidade, a *Open Science* busca tornar a pesquisa em software mais acessível, confiável e relevante para os praticantes da indústria.

A crescente adoção da *Open Science* em diferentes áreas do conhecimento demonstra sua relevância e eficácia como uma abordagem para impulsionar a inovação e o avanço científico. Comunidades científicas ao redor do mundo estão reconhecendo os benefícios da *Open Science* e estão trabalhando para integrar esses princípios em suas práticas de pesquisa e publicação.

No contexto específico dos projetos de software industriais, a *Open Science* oferece oportunidades significativas para melhorar a transparência dos processos de IC, compartilhar dados e resultados relevantes e promover a colaboração entre pesquisadores e profissionais da indústria. Ao adotar os princípios da *Open Science*, os pesquisadores podem contribuir para uma base de conhecimento mais sólida e confiável, informando práticas de engenharia de software mais eficazes e sustentáveis.

Portanto, neste estudo, buscamos investigar os efeitos das más práticas de IC em projetos de software industriais, com o objetivo de contribuir para uma compreensão mais profunda dos desafios enfrentados pela indústria de software e identificar estratégias para mitigar esses desafios. Ao adotar uma abordagem baseada na *Open Science*, buscamos promover a transparência, a replicabilidade e a relevância dos nossos resultados, visando impactar positivamente tanto a pesquisa acadêmica quanto a prática da engenharia de software na indústria.

7.3 Transparência na metodologia

No decorrer da pesquisa, foram desenvolvidos três estudos secundários para responder às três questões de pesquisa desta dissertação. Cada um desses estudos teve suas próprias questões de pesquisa, abordadas com métodos consolidados de pesquisa quantitativa e qualitativa.

Para o primeiro estudo, utilizamos um formulário para coletar as percepções dos

integrantes dos times de desenvolvimento sobre o impacto negativo das más práticas de IC. Também desenvolvemos um *script* para coleta automatizada de uma métrica a partir do servidor de IC. A análise dos dados incluiu técnicas simples de análise quantitativa, como manipulação de planilhas, e análise qualitativa das respostas do questionário.

No segundo estudo, os dados foram coletados por meio de uma ferramenta externa para análise de indicadores de qualidade de software, um *script* para coleta de dados do servidor de IC e um formulário direcionado aos integrantes dos projetos. A análise envolveu comparação de métricas antes e depois da adoção da IC, análise temporal das métricas de qualidade de software e análise qualitativa das respostas do formulário.

No terceiro estudo, desenvolvemos um *script* para coletar automaticamente métricas do servidor de gerenciamento de versões e de IC. Utilizamos uma técnica avançada de análise qualitativa baseada em uma matriz de correlações e conduzimos um grupo focal para coletar dados qualitativos dos membros das equipes de desenvolvimento. A análise dos dados do grupo focal incluiu análise qualitativa das falas dos participantes.

Todos os dados coletados foram armazenados de forma segura e estão disponibilizados para acesso público em um repositório dedicado para isso no GitHub. Da mesma forma, os *scripts* desenvolvidos para a coleta automatizada de dados estão devidamente disponibilizados para garantir a transparência e a replicabilidade deste estudo.

7.4 Dados, materiais e código aberto

A fim de disponibilizar os artefatos gerados em cada estudo, foram criados repositórios dedicados a cada um de forma individual. No entanto, para facilitar o acesso aos artefatos de código-fonte, dados coletados e materiais resultantes das análises de cada estudo desta dissertação, foi criado um repositório central no GitHub que encontra-se disponível publicamente através do seguinte link: https://github.com/ruben-silva-dev/dissertacao-pcomp-ic.

Este repositório central contém os seguintes artefatos:

- **Código-fonte:** Implementações dos *scripts* utilizados para coleta automatizada de dados, análise dos resultados e outras ferramentas desenvolvidas para os estudos.
- **Dados coletados:** Conjuntos de dados utilizados nas análises quantitativas e qualitativas, disponibilizados em formatos acessíveis.
- **Materiais de análise:** Documentos, apresentações ou outras formas de material resultante das análises conduzidas em cada estudo.

Os dados publicados podem servir como base para a comunidade científica replicar os estudos aqui desenvolvidos e estender os achados com relação à influência das más práticas de IC no desenvolvimento de software de forma geral.

Instruções detalhadas sobre como usar os artefatos disponíveis podem ser encontradas no arquivo README.md do repositório. Todos os artefatos estão licenciados sob a Licença *Creative Commons* 4.0 Internacional[1], permitindo sua livre utilização e distribuição, com a possibilidade de contribuições da comunidade para aprimoramento contínuo.

7.5 Desafios e considerações éticas

Neste estudo, foram adotadas várias medidas para garantir que a pesquisa fosse conduzida de maneira ética e respeitasse os direitos e a privacidade dos participantes envolvidos. Abaixo estão algumas considerações éticas relevantes:

- **Anonimização dos Dados Coletados:** Todos os dados coletados durante os estudos foram rigorosamente anonimizados para proteger a privacidade e a confidencialidade dos participantes. Isso incluiu a remoção de quaisquer informações pessoais identificáveis, como nomes, endereços de e-mail, ou quaisquer outros identificadores individuais. A anonimização dos dados foi realizada antes de qualquer análise ou divulgação, garantindo que os participantes permanecessem não identificáveis nos resultados publicados.
- **Consentimento dos Participantes:** Para os estudos que envolveram a coleta de dados diretamente de participantes, todos foram informados sobre os objetivos da pesquisa, os procedimentos envolvidos e quaisquer riscos potenciais associados à sua participação. Os participantes foram convidados a consentir voluntariamente em participar dos estudos e concordar com os termos de uso dos dados coletados.
- **Confidencialidade dos Dados:** Os dados coletados foram tratados com estrita confidencialidade e acessados apenas pela equipe de pesquisa autorizada. Todas as informações pessoais foram armazenadas de forma segura e protegidas contra acesso não autorizado. Apenas os membros da equipe que precisavam acessar os dados para fins de análise e processamento foram autorizados a fazê-lo.
- **Respeito aos Direitos dos Participantes:** Todos os participantes foram tratados com respeito e consideração durante o processo de pesquisa. Suas opiniões e contribuições foram valorizadas e usadas apenas para os fins declarados no consentimento informado.

[1] https://creativecommons.org/licenses/by/4.0/

Qualquer solicitação de retirada ou modificação de dados por parte dos participantes foi prontamente atendida.

– **Cumprimento das Normas Éticas:** Esta pesquisa foi conduzida em conformidade com todas as normas éticas e regulamentações relevantes, incluindo as diretrizes institucionais e as regulamentações legais aplicáveis. Qualquer preocupação ética foi cuidadosamente considerada e abordada durante o planejamento e a execução da pesquisa.

Ao adotar essas medidas éticas, buscou-se garantir a integridade, confiabilidade e respeito pelos direitos dos participantes no estudo sobre os efeitos das más práticas de IC em projetos de *software* industriais.

7.6 Conclusão e recomendações

A pesquisa conduzida neste estudo sobre os efeitos das más práticas de IC em projetos de software industriais oferece *insights* valiosos para a comunidade acadêmica e profissional. Ao adotar os princípios da *Open Science*, buscou-se promover a transparência, replicabilidade e acessibilidade dos resultados, bem como garantir considerações éticas em todo o processo de pesquisa.

Os resultados dos estudos secundários destacaram a importância da IC para a qualidade e eficiência do desenvolvimento de software, mas também apontaram os riscos associados às más práticas de IC. A análise quantitativa e qualitativa revelou impactos negativos significativos nas percepções dos desenvolvedores, na qualidade do código e no desempenho geral dos projetos.

Além disso, a disponibilização dos artefatos de pesquisa, incluindo código-fonte, dados coletados e materiais de análise, em um repositório central no GitHub, permite que outros pesquisadores repliquem e estendam os estudos aqui realizados. Essa prática reforça os princípios da *Open Science* e promove o avanço do conhecimento na área.

Com base nos resultados obtidos, algumas recomendações podem ser feitas para profissionais da indústria de software e pesquisadores interessados em aprimorar os estudos na área de IC:

1. **Adoção de Boas Práticas de IC:** Empresas e equipes de desenvolvimento de software devem priorizar a adoção de boas práticas de IC, como testes automatizados, integração frequente e análise estática de código, para garantir a qualidade e estabilidade dos projetos.
2. **Conscientização sobre os Riscos:** É importante que os profissionais estejam cientes dos

potenciais riscos associados às más práticas de IC, como *builds* quebradas, regressões de código e impacto na produtividade da equipe.

3. **Investimento em Educação e Treinamento:** As empresas devem investir em programas de educação e treinamento para desenvolvedores, a fim de promover o entendimento e a implementação correta de práticas de IC.
4. **Pesquisa Contínua:** A comunidade acadêmica deve continuar a investigar os efeitos das más práticas de IC em diferentes contextos e desenvolver soluções para mitigar seus impactos negativos.

Por fim, é fundamental que todos os envolvidos na pesquisa e prática de IC trabalhem em conjunto para promover um ambiente de desenvolvimento de software mais eficiente, confiável e sustentável. Essas recomendações e conclusões refletem o compromisso deste estudo em contribuir para a compreensão e melhoria contínua da prática de IC em projetos de software industriais.

8 CONCLUSÕES E TRABALHOS FUTUROS

As más práticas de IC podem acarretar problemas para organizações que adotam essa metodologia em seus projetos. Estudos nesse campo alertam para a necessidade de utilizar corretamente os recursos da IC para alcançar resultados desejáveis em termos de qualidade, tanto no processo quanto no produto final (Gallaba; McIntosh, 2020). Alguns autores afirmam que as más práticas de IC podem distorcer esse processo. Isso pode levar as equipes de desenvolvimento a acreditar que estão aplicando corretamente o processo, quando na realidade não o estão fazendo (Felidré *et al.*, 2019).

Apesar da existência dessas más práticas de IC, é fato que o processo é amplamente utilizado por diversas organizações e, em geral, tem gerado resultados positivos (Pinto *et al.*, 2018; Rebouças *et al.*, 2017). Além disso, a IC é uma prática flexível e permite sua integração com outras práticas já consolidadas nos ambientes de desenvolvimento, como é o caso da RC. É importante ressaltar que vários estudos têm investigado a relação entre IC e RC, indicando que a IC exerce principalmente uma influência positiva sobre as atividades de RC (Rahman; Roy, 2017; Zampetti *et al.*, 2019).

Alguns estudos também investigam os efeitos da IC na qualidade de um produto de *software* (Santos *et al.*, 2022; Freitas *et al.*, 2023). Nesse sentido, resultados obtidos em algumas pesquisas indicam que a IC pode influenciar de forma significativa a qualidade de software. Além disso ela pode promover a adoção de boas práticas, como automação de testes, análise estática e aplicação de refatorações, as quais, por sua vez, contribuem para a melhoria da qualidade (Alizadeh *et al.*, 2019; Freitas, 2020; Elazhary *et al.*, 2021).

Apesar dos estudos existentes na literatura, percebemos lacunas e questões não muito exploradas, como o impacto das más práticas de IC na qualidade do *software* e no processo de RC. Assim, esta dissertação busca preencher essas lacunas por meio de um estudo empírico no contexto de projetos industriais, visando aprofundar o conhecimento com relação: (1) as más práticas de IC mais comuns/prejudiciais; (2) a influência das más práticas de IC na qualidade do *software*; e (3) os efeitos das más práticas de IC nas atividades de RC. Este capítulo encerra com considerações finais e delineia as principais contribuições deste trabalho.

8.1 Respondendo às questões de pesquisa

Esta dissertação tem o objetivo principal de investigar os efeitos das más práticas de IC no contexto de projetos industriais. Para tanto, inicialmente foi realizado um estudo para determinar, na percepção das equipes de desenvolvimento, quais as más práticas que mais ocorrem e os seus efeitos de forma geral no contexto de projetos industriais. Este estudo é apresentado no Capítulo 4.

Este estudo inicial permitiu verificar como as más práticas ocorrem e entender os seus efeitos especificamente no contexto de projetos industriais. Além disso, foram identificadas as más práticas de IC consideradas mais prejudiciais para o bom andamento dos projetos. Para analisar a relação das más práticas com a qualidade de *software* bem como a relação dos processos de IC e RC nós implementamos outros dois estudos que, por sua vez, refletem as questões de pesquisa definidas para esta dissertação. A seguir os resultados das questões de pesquisa são discutidas com detalhes.

8.1.0.1 Más práticas de IC mais frequentes e com maior impacto negativo em projetos industriais (QP_1)

Esta questão de pesquisa foi abordada no estudo inicial apresentado no Capítulo 4. Neste estudo foram captadas as percepções de integrantes das equipes de desenvolvimento de projetos industriais acerca da ocorrência e do nível de impacto das más práticas de IC nos projetos em que atuaram. Além disso, foi realizada uma análise do histórico de *pipelines* dos sistemas com o intuito de identificar o mau uso da IC através da detecção de intervalos de correção de *build*. Com isso foi possível obter uma visão preliminar do impacto das más práticas de IC em projetos industriais.

Os resultados indicam que as más práticas mais frequentes estão relacionadas ao gerenciamento de repositório, controle de versão e cultura do ambiente de trabalho, destacando-se o uso de *feature branches* em vez de *feature toggles*, *branches* divergentes e falta de controle total do ambiente pelos desenvolvedores. Além disso, constata-se que essas más práticas têm impacto negativo, sendo que algumas, como *branches* divergentes e ausência de testes em *feature branches* e ambiente de produção, têm forte impacto.

No que diz respeito ao tempo para correção de falhas de *build*, a maioria das falhas não é resolvida imediatamente, podendo levar meses para serem corrigidas. Os dados revelam

que mais de 50% das correções de *build* demoram mais de 3 horas, e as durações variam consideravelmente, com registros que vão desde minutos até meses para correção.

Além disso, as más práticas de IC impactam negativamente a saúde do software, resultando em problemas como aumento de tempo e custo para desenvolver e implantar funcionalidades, propagação de erros, diminuição da produtividade, insegurança na implantação devido a falhas de automação, entre outros. As observações dos participantes indicam uma compreensão ampla dos danos causados pelas más práticas de IC, principalmente relacionados ao desenvolvimento e à garantia da qualidade do software.

8.1.0.2 Impacto das más práticas de IC podem na qualidade do software em projetos industriais (QP_2)

Esta questão de pesquisa foi respondida através de uma análise dos efeitos das más práticas de IC na qualidade do software. Para isso, definimos três subquestões de pesquisa que abordam o impacto das más práticas nos atributos de qualidade interna, a evolução dos indicadores de qualidade ao longo do tempo e a priorização das más práticas para aprimorar a qualidade. O estudo seguiu quatro passos distintos: (1) seleção de nove sistemas de código fechado, (2) coleta e análise dos atributos internos de qualidade antes e depois da implementação da IC, (3) análise do histórico dos indicadores de qualidade e problemas levantados pela análise estática, e (4) desenvolvimento, aplicação e análise de um questionário direcionado às equipes de desenvolvimento dos projetos analisados. Utilizando dados obtidos nessas etapas, o estudo classifica as más práticas de IC de acordo com seu impacto na qualidade do software e o esforço necessário para resolvê-las, resultando em uma lista priorizada de más práticas para correção, visando a melhoria da qualidade do software.

Apresentamos uma análise dos resultados obtidos ao introduzir a IC em cinco projetos selecionados. Ao investigar o impacto nos atributos internos de qualidade, foi observado um aumento nos valores brutos de todos os atributos de qualidade interna para cada sistema após a implementação da IC. Embora seja comum um aumento na complexidade durante o desenvolvimento, a coesão do código também aumentou em todos os casos, sugerindo que a IC pode contribuir para manter a coesão do código. Isso foi corroborado pela análise estática implementada como parte do processo de IC. Concluímos que a IC pode indiretamente melhorar a qualidade do software, especialmente a coesão, evidenciando que o desenvolvimento auxiliado pela IC pode aumentar o grau de coesão dos sistemas.

Na análise da evolução dos indicadores de qualidade, foi realizado um estudo histórico dos indicadores de qualidade e do número de problemas relatados pelo sistema de análise estática (*SonarQube*). Os gráficos mostram uma relação entre os indicadores de qualidade e o número de problemas reportados, indicando que problemas de IC não resolvidos podem prejudicar o nível de qualidade do software. A existência de más práticas de IC na organização abordada no estudo, bem como, estratégias deficientes na definição dos níveis de qualidade dos projetos e o uso de ferramentas de configuração padrão que podem não atender às necessidades dos sistemas são fatores que podem levar à diminuição da qualidade de software.

Além disso, ao priorizar a resolução das más práticas de IC, foi realizada uma análise de correlação entre o impacto e o esforço necessário para resolver essas práticas. A matriz de dificuldade versus impacto destacou a prioridade de resolução das más práticas, indicando que as relacionadas à Cultura, Garantia de Qualidade e Repositório são cruciais para melhorar a qualidade do software. Conclui-se que uma boa arquitetura de teste e a adoção de uma cultura orientada a *DevOps* são práticas recomendadas relacionadas à IC que podem melhorar significativamente a qualidade do software.

8.1.0.3 Relação entre os processos de IC e RC em projetos industriais (QP_3)

Finalmente, para responder à terceira questão de pesquisa nos propomos a analisar a relação entre os processos de IC e RC em projetos industriais. Para isso, utilizamos 32 métricas de IC e RC, além das percepções das equipes de desenvolvimento de dez projetos industriais. Formulamos quatro subquestões de pesquisa na quais (1) investigamos os aspectos correlacionados entre os processos de IC e RC, (2) analisamos as percepções das equipes sobre essas correlações identificadas, (3) exploramos o impacto das más práticas de IC na RC e (4) investigamos os benefícios e desafios do uso conjunto de IC e RC. Para tanto, selecionamos dez projetos de código fechado e coletamos as métricas de IC e RC a partir do repositório de integração contínua utilizado por nossos parceiros na industria. Uma análise estatística, incluindo a normalização de dados, tratamento de *outliers* e o coeficiente de correlação de *Spearman*, foi aplicada para investigar as correlações entre as métricas. Além disso, foi realizado um grupo focal com especialistas que atuaram como revisores de código nos projetos para coletar percepções sobre a influência da IC na RC, efeitos das más práticas de IC na RC, bem como benefícios e desafios da IC na RC. As discussões foram conduzidas utilizando ferramentas virtuais, e os resultados foram analisados com base nas notas compartilhadas no quadro e nas

discussões dos participantes.

Os resultados indicam diversas correlações entre métricas de IC e RC, demonstrando que algumas práticas durante a IC influenciam diretamente a RC. Foram identificadas 23 correlações fortes entre métricas nas matrizes de correlação individuais e na geral, destacando, por exemplo, a relação entre o tempo de execução da IC e várias métricas da RC, como tempo de revisão e quantidade de participantes. A análise sugere que o tempo de IC pode impactar significativamente o tempo de RC. Além disso, as percepções das equipes de desenvolvimento indicaram concordância moderada com afirmações relacionando o aumento do tempo total de revisão proporcional ao tempo total de execução da IC e ao aumento do tempo de revisão com mais pipelines de IC executados. Notavelmente, foi observado que más práticas de IC podem afetar adversamente a RC, com destaque para problemas relacionados a divergências em *branches*, falta de testes em *feature branches*, passos manuais na pipeline e *scripts* de *build* longos, todos gerando atrasos e aumentando a carga de trabalho dos revisores. No entanto, algumas más práticas foram mencionadas apenas uma vez pelos revisores, sugerindo que algumas podem ser mais comuns ou mais impactantes. Por fim, foram destacados benefícios da IC na RC, como identificação proativa de problemas, detecção antecipada de *bugs*, distribuição eficiente de atualizações e redução de tarefas repetitivas, mas também foram apontados desafios, como garantir a qualidade do código e resolver conflitos durante a RC, mesmo com a implementação da IC.

8.2 Publicações

Até o momento da escrita desta dissertação nós publicamos três trabalhos relacionados ao tema. O último trabalho descrito no Capítulo 6 encontra-se em processo de submissão. A Tabela 20 apresenta os dados relacionados às publicações.

Tabela 20 – Publicações realizadas

PUBLICAÇÃO	DESCRIÇÃO
SILVA, R. B. T.; BEZERRA, C. I. M. Analyzing continuous integration bad practices in closed-source projects: An initial study. In:Proceedings of the 34th Brazilian Symposium on Software Engineering. New York, NY, USA: Association for Computing Machinery, 2020. (SBES '20), p. 642–647. ISBN 9781450387538. Disponível em:https://doi.org/silva2020analyzing.	Resultado direto do trabalho desta dissertação. O estudo consistiu em uma investigação acerca da percepção das equipes de desenvolvimento de projetos industriais acerca da existência e do impacto causado pelas más práticas de IC em seus projetos. O estudo é apresentado no Capítulo 4.
SILVA, Ruben Blenicio Tavares; BEZERRA, Carla Ilane Moreira. Investigating the Impact of Bad Practices in Continuous Integration on Closed-source Projects. In: WORKSHOP DE TESES E DISSERTAÇÕES (WTDSOFT) - CONGRESSO BRASILEIRO DE SOFTWARE: TEORIA E PRÁTICA (CBSOFT), 12. , 2021, Joinville. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2021 . p. 46-52. DOI: https://doi.org/10.5753/cbsoft_estendido.2021.17283.	Outro resultado do trabalho desta dissertação. O estudo apresentou um resumo do trabalho desta dissertação com os resultados preliminares bem como a proposta para finalização do objetivo geral da pesquisa. Por se tratar de um trabalho de apresentação preliminar de resultados que ainda não haviam sido consolidados nós optamos por omiti-lo no texto principal desta dissertação.
SILVA, Ruben Blenicio Tavares; BEZERRA, Carla. Empirical investigation of the influence of continuous integration bad practices on software quality. In: WORKSHOP DE VISUALIZAÇÃO, EVOLUÇÃO E MANUTENÇÃO DE SOFTWARE (VEM), 10. , 2022, Online. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2022 . p. 51-55. DOI: https://doi.org/10.5753/vem.2022.226816.	Este trabalho também é resultado direto desta dissertação. O trabalho consistiu em uma análise quantitativa acerca dos impactos das más práticas de IC com relação a qualidade de software. Nesta dissertação, o estudo é apresentado no Capítulo 5.

Fonte: Elaborado pelo autor.

8.3 Trabalhos futuros

A seguir nós listamos alguns possíveis trabalhos futuros com base nos resultados apresentados nesta dissertação:

1. **Investigação mais aprofundada das más práticas de IC**: Nós abordamos algumas más práticas de IC e seus impactos, mas pode ser útil expandir essa pesquisa para identificar outras más práticas específicas, suas causas e efeitos em uma variedade mais ampla de ambientes de desenvolvimento.
2. **Estudo sobre a mitigação de más práticas de IC**: Um estudo mais detalhado sobre como mitigar ou corrigir essas más práticas pode ser valioso. Isso pode incluir o desenvolvimento de diretrizes, boas práticas ou ferramentas específicas para abordar cada má prática identificada.
3. **Aprofundamento na relação entre IC e Qualidade do Software**: Embora a dissertação tenha abordado o impacto das más práticas de IC na qualidade do software, uma investigação mais detalhada sobre como a IC pode influenciar diferentes aspectos da qualidade do software (além dos atributos internos já abordados) seria valiosa.
4. **Estudo sobre a melhoria da relação entre IC e RC**: Uma análise mais detalhada dos métodos ou estratégias para aprimorar a RC em ambientes que também utilizam a IC pode fornecer *insights* sobre como melhorar a eficiência e a eficácia do processo de RC nesses contextos.
5. **Exploração de soluções para mitigar impactos negativos das más práticas de IC na RC**: Investigar estratégias ou ferramentas que possam ajudar a minimizar os impactos adversos das más práticas de IC na RC pode ser um campo promissor de pesquisa.
6. **Estudo sobre a automação na IC e seu impacto na RC**: Uma análise mais aprofundada sobre como a automação dentro da IC pode influenciar ou melhorar o processo de RC pode ser um ponto de interesse, especialmente explorando a eficácia de diferentes níveis de automação.
7. **Análise de boas práticas e desafios em ambientes de IC e RC**: Investigar em profundidade as boas práticas que podem ser implementadas para melhorar a integração entre IC e RC, assim como os desafios enfrentados na adoção dessas práticas, pode oferecer orientações práticas para equipes de desenvolvimento.
8. **Avaliação de ferramentas e tecnologias emergentes**: Explorar o uso de tecnologias emergentes, como IA/ML, para otimizar processos de IC e RC e avaliar como essas

ferramentas podem ajudar a identificar e corrigir más práticas de forma mais eficiente.

Essas direções de pesquisa podem contribuir significativamente para o aprimoramento da compreensão e práticas relacionadas à IC e RC, resultando em melhorias tangíveis na qualidade do software e nos processos de desenvolvimento de software em geral.

REFERÊNCIAS

ACKERMAN, A.; BUCHWALD, L.; LEWSKI, F. Software inspections: an effective verification process. **IEEE Software**, v. 6, n. 3, p. 31–36, 1989.

ACKERMAN, A. F.; FOWLER, P. J.; EBENAU, R. G. Software inspections and the industrial production of software. In: **Proc. of a Symposium on Software Validation**: Inspection-testing-verification-alternatives. USA: Elsevier North-Holland, Inc., 1984. p. 13–40. ISBN 044487593X.

ALIZADEH, V.; OUALI, M. A.; KESSENTINI, M.; CHATER, M. Refbot: Intelligent software refactoring bot. In: **2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)**. [*S. l.*: *s. n.*], 2019. p. 823–834.

ALMEIDA, C.; KALINOWSKI, M.; UCHôA, A.; FEIJó, B. Negative effects of gamification in education software: Systematic mapping and practitioner perceptions. **Information and Software Technology**, v. 156, p. 107142, 2023. ISSN 0950-5849. Disponível em: https://www.sciencedirect.com/science/article/pii/S0950584922002518. Acesso em: 25 abr. 2024.

BACCHELLI, A.; BIRD, C. Expectations, outcomes, and challenges of modern code review. In: **2013 35th International Conference on Software Engineering (ICSE)**. [*S. l.*: *s. n.*], 2013. p. 712–721.

BAUM, T.; LISKIN, O.; NIKLAS, K.; SCHNEIDER, K. A faceted classification scheme for change-based industrial code review processes. In: **2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)**. [*S. l.*: *s. n.*], 2016. p. 74–85.

BELLER, M.; GOUSIOS, G.; ZAIDMAN, A. Oops, my tests broke the build: An explorative analysis of travis ci with github. In: IEEE. **2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)**. [*S. l.*], 2017.

BIBIANO, A. C.; SOARES, V.; COUTINHO, D.; FERNANDES, E.; CORREIA, J. a. L.; SANTOS, K.; OLIVEIRA, A.; GARCIA, A.; GHEYI, R.; FONSECA, B.; RIBEIRO, M.; BARBOSA, C.; OLIVEIRA, D. How does incomplete composite refactoring affect internal quality attributes? In: **Proceedings of the 28th International Conference on Program Comprehension**. New York, NY, USA: Association for Computing Machinery, 2020. (ICPC '20), p. 149–159. ISBN 9781450379588. Disponível em: https://doi.org/10.1145/3387904.3389264. Acesso em: 25 abr. 2024.

BISONG, E. Introduction to scikit-learn. In: ____. **Building Machine Learning and Deep Learning Models on Google Cloud Platform**: A comprehensive guide for beginners. Berkeley, CA: Apress, 2019. p. 215–229. ISBN 978-1-4842-4470-8. Disponível em: https://doi.org/10.1007/978-1-4842-4470-8_18. Acesso em: 25 abr. 2024.

BOOCH, G. **Object oriented design with applications**. [*S. l.*]: Benjamin-Cummings Publishing Co., Inc., 1990.

BROOKS, F. **The Mythical Man-Month**: Essays on softw. 1st. [*S. l.*]: Addison-Wesley Longman Publishing Co., Inc, 1978.

CANO, P. O.; MEJIA, A. M.; AVILA, S. D. G.; DOMINGUEZ, G. E. Z.; MORENO, I. S.; LEPE, A. N. A taxonomy on continuous integration and deployment tools and frameworks. In: MEJIA, J.; MUÑOZ, M.; ROCHA, Á.; QUIÑONEZ, Y. (Ed.). **New Perspectives in Software Engineering**. Cham: Springer International Publishing, 2021. p. 323–336. ISBN 978-3-030-63329-5.

CASSEE, N.; VASILESCU, B.; SEREBRENIK, A. The silent helper: The impact of continuous integration on code reviews. In: **2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)**. [*S. l.*: *s. n.*], 2020. p. 423–434.

CHEN, D.; STOLEE, K. T.; MENZIES, T. Replication can improve prior results: A github study of pull request acceptance. In: **2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)**. [*S. l.*: *s. n.*], 2019. p. 179–190.

CHIDAMBER, S. R.; KEMERER, C. F. A metrics suite for object oriented design. **IEEE Transactions on software engineering**, IEEE, v. 20, n. 6, p. 476–493, 1994.

Dalla Palma, S.; Di Nucci, D.; PALOMBA, F.; TAMBURRI, D. A. Toward a catalog of software quality metrics for infrastructure code. **Journal of Systems and Software**, v. 170, p. 110726, 2020. ISSN 0164-1212. Disponível em: https://www.sciencedirect.com/science/article/pii/S0164121220301618. Acesso em: 25 abr. 2024.

DALLAL, J. A. Object-oriented class maintainability prediction using internal quality attributes. **Inf. Softw. Technol.**, Elsevier, v. 55, n. 11, p. 2028–2048, 2013.

DEBBICHE, A.; DIENÉR, M.; SVENSSON, R. B. Challenges when adopting continuous integration: A case study. In: JEDLITSCHKA, A.; KUVAJA, P.; KUHRMANN, M.; MÄNNISTÖ, T.; MÜNCH, J.; RAATIKAINEN, M. (Ed.). **Product-Focused Software Process Improvement**. Cham: Springer International Publishing, 2014. p. 17–32. ISBN 978-3-319-13835-0.

DESTEFANIS, G.; COUNSELL, S.; CONCAS, G.; TONELLI, R. Software metrics in agile software: An empirical study. In: SPRINGER. **International Conference on Agile Software Development**. [*S. l.*], 2014. p. 157–170.

DOWNS, J.; HOSKING, J.; PLIMMER, B. Status communication in agile software teams: A case study. In: IEEE. **2010 Fifth International Conference on Software Engineering Advances**. [*S. l.*], 2010.

DRISCOLL, W. C. Robustness of the anova and tukey-kramer statistical tests. **Computers & Industrial Engineering**, Elsevier, v. 31, n. 1-2, p. 265–268, 1996.

DUVALL, P. **Continuous Delivery Patterns and AntiPatterns in the Software LifeCycle**. 2018. Disponível em: https://dzone.com/refcardz/continuous-delivery-patterns. Acesso em: 25 abr. 2024.

DUVALL, P. M. **Continuous Integration**: Patterns and anti-patterns. [*S. l.*]: DZone, Incorporated, 2010.

ELAZHARY, O.; WERNER, C.; LI, Z. S.; LOWLIND, D.; ERNST, N. A.; STOREY, M.-A. Uncovering the benefits and challenges of continuous integration practices. **IEEE Transactions on Software Engineering**, 2021.

FAGAN, M. E. Design and code inspections to reduce errors in program development. **IBM Systems Journal**, v. 15, n. 3, p. 182–211, 1976.

FATIMA, N.; CHUPRAT, S.; NAZIR, S. Challenges and benefits of modern code review-systematic literature review protocol. In: **2018 International Conference on Smart Computing and Electronic Enterprise (ICSCEE)**. [*S. l.*: *s. n.*], 2018. p. 1–5.

FELIDRÉ, W.; FURTADO, L.; COSTA, D. A. da; CARTAXO, B.; PINTO, G. Continuous integration theater. In: **2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)**. [*S. l.*: *s. n.*], 2019.

FERNANDES, E.; CHáVEZ, A.; GARCIA, A.; FERREIRA, I.; CEDRIM, D.; SOUSA, L.; OIZUMI, W. Refactoring effect on internal quality attributes: What haven't they told you yet? **Information and Software Technology**, v. 126, p. 106347, 2020. ISSN 0950-5849. Disponível em: https://www.sciencedirect.com/science/article/pii/S0950584920301142. Acesso em: 25 abr. 2024.

FITZGERALD, B.; STOL, K.-J. Continuous software engineering: A roadmap and agenda. **Journal of Systems and Software**, v. 123, p. 176–189, 2017. ISSN 0164-1212. Disponível em: https://www.sciencedirect.com/science/article/pii/S0164121215001430. Acesso em: 25 abr. 2024.

FOWLER, M. **Continuous integration**. 2006. Disponível em: https://martinfowler.com/articles/continuousIntegration.html. Acesso em: 25 abr. 2024.

FOWLER, M. **Refactoring**: improving the design of existing code. [*S. l.*]: Addison-Wesley Professional, 2018.

FREITAS, G.; BERNARDO, J. a. H.; SIZÍLIO, G.; COSTA, D. A. D.; KULESZA, U. Analyzing the impact of ci sub-practices on continuous code quality in open-source projects: An empirical study. In: **Proceedings of the XXXVII Brazilian Symposium on Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2023. (SBES '23), p. 1–10. ISBN 9798400707872. Disponível em: https://doi.org/10.1145/3613372.3613403. Acesso em: 25 abr. 2024.

FREITAS, G. D. D. d. **Investigating the relationship between continuous integration and software quality metrics**: an empirical study. 59 f. Dissertação (Mestrado em Sistemas e Computação) – Centro de Ciências Exatas e da Terra, Universidade Federal do Rio Grande do Norte, Natal, 2020.

GALLABA, K.; MCINTOSH, S. Use and misuse of continuous integration features: An empirical study of projects that (mis)use travis ci. **IEEE Transactions on Software Engineering**, v. 46, n. 1, p. 33–50, 2020.

GHALEB, T. A.; COSTA, D. A. D.; ZOU, Y. An empirical study of the long duration of continuous integration builds. **Empirical Software Engineering**, Springer, 2019.

GOODMAN, D.; ELBAZ, M. "it's not the pants, it's the people in the pants"learnings from the gap agile transformation what worked, how we did it, and what still puzzles us. In: IEEE. **Agile 2008 Conference**. [*S. l.*], 2008.

GOUSIOS, G.; PINZGER, M.; DEURSEN, A. v. An exploratory study of the pull-based software development model. In: **Proceedings of the 36th International Conference on Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2014. (ICSE 2014), p. 345–355. ISBN 9781450327565. Disponível em: https://doi.org/10.1145/2568225.2568260. Acesso em: 25 abr. 2024.

GOUSIOS, G.; ZAIDMAN, A. A dataset for pull-based development research. In: **Proceedings of the 11th Working Conference on Mining Software Repositories**. New York, NY, USA: Association for Computing Machinery, 2014. (MSR 2014), p. 368–371. ISBN 9781450328630. Disponível em: https://doi.org/10.1145/2597073.2597122. Acesso em: 25 abr. 2024.

GUERRIERO, M.; GARRIGA, M.; TAMBURRI, D. A.; PALOMBA, F. Adoption, support, and challenges of infrastructure-as-code: Insights from industry. In: **2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)**. [*S. l.*: *s. n.*], 2019. p. 580–589.

HAMDAN, S.; ALRAMOUNI, S. A quality framework for software continuous integration. **Procedia Manufacturing**, v. 3, p. 2019–2025, 2015. ISSN 2351-9789. 6th International Conference on Applied Human Factors and Ergonomics (AHFE 2015) and the Affiliated Conferences, AHFE 2015. Disponível em: https://www.sciencedirect.com/science/article/pii/S2351978915002504. Acesso em: 25 abr. 2024.

HAMDI, O.; OUNI, A.; ALOMAR, E. A.; CINNéIDE, M. ; MKAOUER, M. W. An empirical study on the impact of refactoring on quality metrics in android applications. In: **2021 IEEE/ACM 8th International Conference on Mobile Software Engineering and Systems (MobileSoft)**. [*S. l.*: *s. n.*], 2021. p. 28–39.

HILTON, M.; NELSON, N.; TUNNELL, T.; MARINOV, D.; DIG, D. Trade-offs in continuous integration: Assurance, security, and flexibility. In: **Proceedings of the 11th Joint Meeting on Foundations of Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2017. (ESEC/FSE 2017), p. 197–207. ISBN 9781450351058. Disponível em: https://doi.org/10.1145/3106237.3106270. Acesso em: 25 abr. 2024.

HILTON, M.; TUNNELL, T.; HUANG, K.; MARINOV, D.; DIG, D. Usage, costs, and benefits of continuous integration in open-source projects. In: **Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2016. (ASE 2016), p. 426–437. ISBN 9781450338455. Disponível em: https://doi.org/10.1145/2970276.2970358. Acesso em: 25 abr. 2024.

JASPAN, C.; JORDE, M.; KNIGHT, A.; SADOWSKI, C.; SMITH, E. K.; WINTER, C.; MURPHY-HILL, E. Advantages and disadvantages of a monolithic repository: A case study at google. In: **Proceedings of the 40th International Conference on Software Engineering**: Software engineering in practice. New York, NY, USA: Association for Computing Machinery, 2018. (ICSE-SEIP '18), p. 225–234. ISBN 9781450356596. Disponível em: https://doi.org/10.1145/3183519.3183550. Acesso em: 25 abr. 2024.

JIANG, J.; LV, J.; ZHENG, J.; ZHANG, L. How developers modify pull requests in code review. **IEEE Transactions on Reliability**, p. 1–15, 2021.

KITCHENHAM, B. A. Software quality assurance. **Microprocessors and Microsystems**, v. 13, n. 6, p. 373–381, 1989. ISSN 0141-9331. Disponível em: https://www.sciencedirect.com/science/article/pii/0141933189900458. Acesso em: 25 abr. 2024.

LAI, S.-T.; SUSANTO, H.; LEU, F.-Y. Combining pipeline quality with automation to ci/cd process for improving quality and efficiency of software maintenance. In: BAROLLI, L.; YIM, K.; CHEN, H.-C. (Ed.). **Innovative Mobile and Internet Services in Ubiquitous Computing**. Cham: Springer International Publishing, 2021. p. 362–372. ISBN 978-3-030-79728-7.

LAUKKANEN, E.; ITKONEN, J.; LASSENIUS, C. Problems, causes and solutions when adopting continuous delivery—a systematic literature review. **Information and Software Technology**, v. 82, p. 55–79, 2017. ISSN 0950-5849. Disponível em: https://www.sciencedirect.com/science/article/pii/S0950584916302324. Acesso em: 25 abr. 2024.

Laukkanen, E.; Paasivaara, M.; Arvonen, T. Stakeholder perceptions of the adoption of continuous integration – a case study. In: **2015 Agile Conference**. [*S. l.*: *s. n.*], 2015. p. 11–20.

LORENZ, M.; KIDD, J. **Object-oriented software metrics**: a practical guide. [*S. l.*]: Prentice-Hall, Inc., 1994.

MACLEOD, L.; GREILER, M.; STOREY, M.-A.; BIRD, C.; CZERWONKA, J. Code reviewing in the trenches: Challenges and best practices. **IEEE Software**, v. 35, n. 4, p. 34–42, 2018.

MALHOTRA, R. **Empirical research in software engineering**: concepts, analysis, and applications. [*S. l.*]: Chapman and Hall/CRC, 2019.

MALHOTRA, R.; CHUG, A. An empirical study to assess the effects of refactoring on software maintainability. In: IEEE. **International Conference on Advances in Computing, Communications and Informatics (ICACCI)**. [*S. l.*], 2016. p. 110–117.

MARTINS, J.; BEZERRA, C.; UCHôA, A.; GARCIA, A. Are code smell co-occurrences harmful to internal quality attributes? a mixed-method study. In: **Proceedings of the 34th Brazilian Symposium on Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2020. (SBES '20), p. 52–61. ISBN 9781450387538. Disponível em: https://doi.org/10.1145/3422392.3422419. Acesso em: 25 abr. 2024.

MCCABE, T. J. A complexity measure. **IEEE Trans. Softw. Eng.**, IEEE, n. 4, p. 308–320, 1976.

MCINTOSH, S.; KAMEI, Y.; ADAMS, B.; HASSAN, A. E. The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In: **Proceedings of the 11th Working Conference on Mining Software Repositories**. New York, NY, USA: Association for Computing Machinery, 2014. (MSR 2014), p. 192–201. ISBN 9781450328630. Disponível em: https://doi.org/10.1145/2597073.2597076. Acesso em: 25 abr. 2024.

MCINTOSH, S.; KAMEI, Y.; ADAMS, B.; HASSAN, A. E. An empirical study of the impact of modern code review practices on software quality. **Empirical Software Engineering**, Springer, v. 21, n. 5, p. 2146–2189, 2016.

MENS, T.; TOURWE, T. A survey of software refactoring. **IEEE Transactions on Software Engineering**, v. 30, n. 2, p. 126–139, 2004.

MEYER, M. Continuous integration and its tools. **IEEE Software**, v. 31, n. 3, p. 14–16, 2014.

MILLER, A. A hundred days of continuous integration. In: IEEE. **Agile 2008 conference**. [*S. l.*], 2008.

MLADENOVA, T. Software quality metrics – research, analysis and recommendation. In: **2020 International Conference Automatics and Informatics (ICAI)**. [*S. l.*: *s. n.*], 2020. p. 1–5.

MOHAMMAD, S. M. Continuous integration and automation. **International Journal of Creative Research Thoughts (IJCRT), ISSN**, p. 2320–2882, 2016.

MORASCA, S. A probability-based approach for measuring external attributes of software artifacts. In: IEEE COMPUTER SOCIETY. **3rd ESEM**. [*S. l.*], 2009. p. 44–55.

OLSSON, H. H.; ALAHYARI, H.; BOSCH, J. 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**. [*S. l.*: *s. n.*], 2012. p. 392–399.

PAIXAO, M.; KRINKE, J.; HAN, D.; HARMAN, M. Crop: Linking code reviews to source code changes. In: **Proceedings of the 15th International Conference on Mining Software Repositories**. New York, NY, USA: Association for Computing Machinery, 2018. (MSR '18), p. 46–49. ISBN 9781450357166. Disponível em: https://doi.org/10.1145/3196398.3196466. Acesso em: 25 abr. 2024.

PHAM, A.; PHAM, P.-V. **SCRUM em ação**. [*S. l.*]: Novatec Editora, 2011.

PINTO, G.; CASTOR, F.; BONIFACIO, R.; REBOUÇAS, M. Work practices and challenges in continuous integration: A survey with travis ci users. **Software: Practice and Experience**, Wiley Online Library, 2018.

RAHMAN, M. M.; ROY, C. K. Impact of continuous integration on code reviews. In: **2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)**. [*S. l.*: *s. n.*], 2017. p. 499–502.

REBOUÇAS, M.; SANTOS, R. O.; PINTO, G.; CASTOR, F. How does contributors' involvement influence the build status of an open-source software project? In: **2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)**. [*S. l.*: *s. n.*], 2017.

RIGBY, P. C.; BIRD, C. Convergent contemporary software peer review practices. In: **Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2013. (ESEC/FSE 2013), p. 202–212. ISBN 9781450322379. Disponível em: https://doi.org/10.1145/2491411.2491444. Acesso em: 25 abr. 2024.

RIGBY, P. C.; GERMAN, D. M.; COWEN, L.; STOREY, M.-A. Peer review on open-source software projects: Parameters, statistical models, and theory. **ACM Trans. Softw. Eng. Methodol.**, Association for Computing Machinery, New York, NY, USA, v. 23, n. 4, set. 2014. ISSN 1049-331X. Disponível em: https://doi.org/10.1145/2594458. Acesso em: 25 abr. 2024.

SAIDANI, I.; OUNI, A.; MKAOUER, M. W.; PALOMBA, F. On the impact of continuous integration on refactoring practice: An exploratory study on travistorrent. **Information and Software Technology**, v. 138, p. 106618, 2021. ISSN 0950-5849. Disponível em: https://www.sciencedirect.com/science/article/pii/S0950584921000914. Acesso em: 25 abr. 2024.

SAINI, N.; BRITTO, R. Using machine intelligence to prioritise code review requests. In: **Proceedings of the 43rd International Conference on Software Engineering**: Software engineering in practice (icse-seip). [*S. l.*: *s. n.*], 2021. p. 11–20.

SANTOS, J.; COSTA, D. Alencar da; KULESZA, U. Investigating the impact of continuous integration practices on the productivity and quality of open-source projects. In: **Proceedings of the 16th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement**. New York, NY, USA: Association for Computing Machinery, 2022. (ESEM '22), p. 137–147. ISBN 9781450394277. Disponível em: https://doi.org/10.1145/3544902.3546244. Acesso em: 25 abr. 2024.

SARAIVA, D.; COSTA, D. A. D.; KULESZA, U.; SIZíLIO, G.; NETO, J. G.; COELHO, R.; NAGAPPAN, M. Unveiling the relationship between continuous integration and code coverage. In: **2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)**. [*S. l.*: *s. n.*], 2023. p. 247–259.

SCHOBER, P.; BOER, C.; SCHWARTE, L. A. Correlation coefficients: appropriate use and interpretation. **Anesthesia & Analgesia**, Wolters Kluwer, [*S. l.*], v. 126, n. 5, p. 1763–1768, 2018.

SHAHIN, M.; BABAR, M. A.; ZHU, L. Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. **IEEE Access**, 2017.

SHI, A.; ZHAO, P.; MARINOV, D. Understanding and improving regression test selection in continuous integration. In: **2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)**. [*S. l.*: *s. n.*], 2019. p. 228–238.

SHULL, F.; SEAMAN, C. Inspecting the history of inspections: An example of evidence-based technology diffusion. **IEEE Software**, v. 25, n. 1, p. 88–90, 2008.

SILVA, R.; BEZERRA, C. Empirical investigation of the influence of continuous integration bad practices on software quality. In: **Anais do X Workshop de Visualização, Evolução e Manutenção de Software**. Porto Alegre, RS, Brasil: SBC, 2022. p. 51–55. ISSN 0000-0000. Disponível em: https://sol.sbc.org.br/index.php/vem/article/view/22330. Acesso em: 25 abr. 2024.

SILVA, R. B. T.; BEZERRA, C. I. M. Analyzing continuous integration bad practices in closed-source projects: An initial study. In: **Proceedings of the 34th Brazilian Symposium on Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2020. (SBES '20), p. 642–647. ISBN 9781450387538. Disponível em: https://doi.org/10.1145/3422392.3422474. Acesso em: 25 abr. 2024.

SINGH, C.; GABA, N. S.; KAUR, M.; KAUR, B. Comparison of different ci/cd tools integrated with cloud platform. In: **2019 9th International Conference on Cloud Computing, Data Science Engineering (Confluence)**. [*S. l.*: *s. n.*], 2019. p. 7–12.

SOARES, E.; COSTA, D. A. da; KULESZA, U. **Continuous Integration and Software Quality**: A causal explanatory study. 2023.

SOARES, E.; SIZILIO, G.; SANTOS, J.; COSTA, D. A. da; KULESZA, U. The effects of continuous integration on software development: a systematic literature review. **Empirical Software Engineering**, v. 27, n. 3, p. 78, Mar 2022. ISSN 1573-7616. Disponível em: https://doi.org/10.1007/s10664-021-10114-1. Acesso em: 25 abr. 2024.

STÅHL, D.; BOSCH, J. Modeling continuous integration practice differences in industry software development. **Journal of Systems and Software**, Elsevier, 2014.

STÅHL, D.; MÅRTENSSON, T.; BOSCH, J. The continuity of continuous integration: Correlations and consequences. **Journal of Systems and Software**, Elsevier, 2017.

THONGTANUNAM, P.; MCINTOSH, S.; HASSAN, A. E.; IIDA, H. 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**. New York, NY, USA: Association for Computing Machinery, 2016. p. 1039–1050. ISBN 9781450339001. Disponível em: https://doi.org/10.1145/2884781.2884852. Acesso em: 25 abr. 2024.

TSAY, J.; DABBISH, L.; HERBSLEB, J. Influence of social and technical factors for evaluating contribution in github. In: **Proceedings of the 36th International Conference on Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2014. (ICSE 2014), p. 356–366. ISBN 9781450327565. Disponível em: https://doi.org/10.1145/2568225.2568315. Acesso em: 25 abr. 2024.

TSOTSIS, A. **Meet phabricator, the witty code review tool built inside facebook**. 2011.

UCHôA, A.; BARBOSA, C.; OIZUMI, W.; BLENíLIO, P.; LIMA, R.; GARCIA, A.; BEZERRA, C. How does modern code review impact software design degradation? an in-depth empirical study. In: **36th ICSME**. [*S. l.*: *s. n.*], 2020.

VALENTE, M. T. **Engenharia de Software Moderna**: Princípios e práticas para desenvolvimento de software com produtividade. [*S. l.*: *s. n.*], 2020.

VASILESCU, B.; YU, Y.; WANG, H.; DEVANBU, P.; FILKOV, V. Quality and productivity outcomes relating to continuous integration in github. In: **Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering**. [*S. l.*: *s. n.*], 2015.

VASSALLO, C.; PALOMBA, F.; BACCHELLI, A.; GALL, H. C. Continuous code quality: are we (really) doing that? In: **Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering**. [*S. l.*: *s. n.*], 2018.

VASSALLO, C.; PROKSCH, S.; GALL, H. C.; PENTA, M. D. Automated reporting of anti-patterns and decay in continuous integration. In: **2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)**. [*S. l.*: *s. n.*], 2019. p. 105–115.

WESSEL, M.; SEREBRENIK, A.; WIESE, I.; STEINMACHER, I.; GEROSA, M. A. Effects of adopting code review bots on pull requests to oss projects. In: **2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)**. [*S. l.*: *s. n.*], 2020. p. 1–11.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. **Experimentation in software engineering**. [*S. l.*]: Springer Science & Business Media, 2012.

YU, Y.; WANG, H.; FILKOV, V.; DEVANBU, P.; VASILESCU, B. Wait for it: Determinants of pull request evaluation latency on github. In: **2015 IEEE/ACM 12th Working Conference on Mining Software Repositories**. [*S. l.*: *s. n.*], 2015. p. 367–371.

ZAMPETTI, F.; BAVOTA, G.; CANFORA, G.; PENTA, M. D. 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)**. [*S. l.*: *s. n.*], 2019. p. 38–48.

ZAMPETTI, F.; SCALABRINO, S.; OLIVETO, R.; CANFORA, G.; PENTA, M. D. 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)**. [*S. l.*: *s. n.*], 2017. p. 334–344.

ZAMPETTI, F.; VASSALLO, C.; PANICHELLA, S.; CANFORA, G.; GALL, H.; PENTA, M. D. An empirical characterization of bad practices in continuous integration. **Empirical Software Engineering**, Springer, 2020.

ZHANG, Y.; YIN, G.; YU, Y.; WANG, H. Investigating social media in github's pull-requests: A case study on ruby on rails. In: **Proceedings of the 1st International Workshop on Crowd-Based Software Development Methods and Technologies**. New York, NY, USA: Association for Computing Machinery, 2014. (CrowdSoft 2014), p. 37–41. ISBN 9781450332248. Disponível em: https://doi.org/10.1145/2666539.2666572. Acesso em: 25 abr. 2024.

APÊNDICE A – FORMULÁRIO DO PRIMEIRO ESTUDO

Formulário - Más práticas de Integração Contínua

IDENTIFICAÇÃO

E-mail:	
Sistema: ( )S1 ( )S2	Papel: ( )Supervisor ( )Desenvolvedor
Tempo de experiência com desenvolvimento de software (em anos):	
Nível de escolaridade: ( )Graduação ( )Mestrado ( )Doutorado	
Nível de conhecimento em integração contínua:	( )Básico ( )Intermediário ( )Avançado

1/7 - REPOSITÓRIO

Para cada má prática, assinale a alternativa correspondente: 1 - Não sei informar, 2 - não existe no projeto, 3 - existe, mas não tem impacto, 4 - existe e tem um impacto leve, 5 - existe e tem um impacto forte.

MÁ PRÁTICA	1	2	3	4	5
Decomposição do projeto no repositório não segue princípios de modularização					
Casos de teste não estão organizados em pastas com base em seus propósitos					
Espaço de trabalho local e remoto não estão alinhados					
Número de branches não se adequa às necessidades/características do projeto					
Falta uma branch de lançamento estável					
Branches de funcionalidades são usadas em vez de toggles de funcionalidades					
Branches divergentes					
Artefatos gerados estão versionados, quando não deveriam estar					
Blobs são desnecessariamente verificados a cada build em vez de serem armazenados em cache					
Recursos relacionados ao pipeline não estão versionados					

2/7 - ESCOLHAS DE INFRAESTRUTURA

Para cada má prática, assinale a alternativa correspondente: 1 - Não sei informar, 2 - não existe no projeto, 3 - existe, mas não tem impacto, 4 - existe e tem um impacto leve, 5 - existe e tem um impacto forte.

MÁ PRÁTICA	1	2	3	4	5
Recursos relacionados à mesma etapa do pipeline estão distribuídos em vários servidores					
O hardware do servidor CI é usado para diferentes fins além de executar o framework CI					
Ferramentas externas são usadas com suas configurações padrão					
Diferentes versões de ferramentas/plugins estão instaladas no mesmo servidor					
Diferentes plugins são usados para realizar a mesma tarefa no mesmo processo de build					
Uma tarefa é implementada usando uma ferramenta/plugin inadequado					
Usar scripts shell para uma tarefa para a qual existe um plugin adequado disponível					

3/7 - ORGANIZAÇÃO DO PROCESSO DE BUILD

Para cada má prática, assinale a alternativa correspondente: 1 - Não sei informar, 2 - não existe no projeto, 3 - existe, mas não tem impacto, 4 - existe e tem um impacto leve, 5 - existe e tem um impacto forte.

MÁ PRÁTICA	1	2	3	4	5
Estratégia inadequada de limpeza do ambiente de build					
Gestão de pacotes ausente					
São usados trabalhos de build amplos e incoerentes					
Builds monolíticos são usados no pipeline					
Trabalhos de build independentes não são executados em paralelo					
Apenas o último commit é construído, abortando builds obsoletas e enfileiradas					
Os passos de build não estão corretamente ordenados					
Etapas/estágios do pipeline são pulados arbitrariamente					
Tarefas não estão distribuídas adequadamente entre diferentes estágios de build					
Builds incrementais são usados sem nunca construir o projeto inteiro do zero					

Estratégia fraca de acionamento de builds					
Builds privados não são usados					
Algumas tarefas do pipeline são iniciadas manualmente					
Uso de builds noturnos					
Projetos inativos estão sendo consultados					
Um build é considerado bem-sucedido quando uma tarefa falha ou ocorre um erro					
Um build falha devido a alguma instabilidade na execução, quando não deveria					
Gestão de dependências não é utilizada					
Inclusão de dependências desnecessárias					
Algumas tarefas são executadas sem relatar claramente seus resultados na saída do build					
A saída de diferentes tarefas de build é misturada na saída do build					
Notificações de falhas são enviadas apenas para equipes/desenvolvedores que se inscreveram explicitamente					
Mecanismo de notificação ausente					
Relatórios de build contêm informações verbosas e irrelevantes					
Tempo limite não está configurado corretamente					
Tarefas não necessárias são agendadas no processo de build					
Tempo de build para o estágio de commit ultrapassa a regra dos 10 minutos					
Passos de reconstrução desnecessários são realizados					
Dados de autenticação são codificados (em claro) no VCS					

4/7 - MANUTENIBILIDADE DO BUILD

Para cada má prática, assinale a alternativa correspondente: 1 - Não sei informar, 2 - não existe no projeto, 3 - existe, mas não tem impacto, 4 - existe e tem um impacto leve, 5 - existe e tem um impacto forte.

MÁ PRÁTICA	**1**	**2**	**3**	**4**	**5**
São usados caminhos absolutos/dependentes da máquina					
Scripts de build são altamente dependentes do IDE					
Variáveis de ambiente não são usadas					
Configurações de build são clonadas nos diferentes ambientes					
Trabalhos de build não são parametrizados					
Scripts de build extensos					

Falta de teste de fumaça, conjunto de testes para verificar a testabilidade do build					
Convenção de nomes rigorosa ausente/pobre para trabalhos de build					

5/7 - GARANTIA DA QUALIDADE

Para cada má prática, assinale a alternativa correspondente: 1 - Não sei informar, 2 - não existe no projeto, 3 - existe, mas não tem impacto, 4 - existe e tem um impacto leve, 5 - existe e tem um impacto forte.

MÁ PRÁTICA	1	2	3	4	5
Falta de testes em um ambiente semelhante ao de produção					
Ferramentas de cobertura de código são executadas apenas durante testes diferentes de unitários e de integração					
Limites de cobertura são fixados com base no que foi alcançado em builds anteriores					
Limites de cobertura são muito altos					
Testes ausentes em branches de funcionalidades					
Todas as permutações de toggles de funcionalidades são testadas					
Recursos de produção são usados para fins de teste					
Testes não são totalmente automatizados, levando a um build não reprodutível					
Suíte de teste contém testes instáveis					
Má escolha no conjunto de casos de teste a serem executados no servidor CI					
Testes falhados são re-executados no mesmo build					
Portões de qualidade são definidos sem os desenvolvedores considerarem apenas o que é ditado pelo cliente					
Usar portões de qualidade para monitorar a atividade de desenvolvedores específicos					
Verificações de análise estática desnecessárias estão incluídas no processo de build					

6/7 - PROCESSO DE ENTREGA

Para cada má prática, assinale a alternativa correspondente: 1 - Não sei informar, 2 - não existe no projeto, 3 - existe, mas não tem impacto, 4 - existe e tem um impacto leve, 5 - existe e tem um impacto forte.

MÁ PRÁTICA	1	2	3	4	5
Artefatos gerados localmente são implantados					
Repositório de artefatos ausente					
Estratégia de rollback ausente					
Estratégia de tag de lançamento ausente					
Verificação de entregáveis ausente					

7/7 - CULTURA

Para cada má prática, assinale a alternativa correspondente: 1 - Não sei informar, 2 - não existe no projeto, 3 - existe, mas não tem impacto, 4 - existe e tem um impacto leve, 5 - existe e tem um impacto forte.

MÁ PRÁTICA	1	2	3	4	5
Mudanças são puxadas antes de corrigir uma falha de build anterior					
Reunião/discussão em equipe é realizada logo antes de enviar para o branch principal					
Desenvolvedores e operadores são mantidos como papéis separados					
Desenvolvedores não têm controle completo do ambiente					
Falhas de build não são corrigidas imediatamente, dando prioridade a outras mudanças					
Notificações de problemas são ignoradas					

AVALIAÇÃO NARRATIVA

Cite e descreva sua percepção dos impactos positivos das boas práticas de CI em relação à saúde do software. Exemplifique para clareza de sua resposta.
Cite e descreva sua percepção dos impactos negativos das más práticas de CI em relação à saúde do software. Exemplifique para clareza de sua resposta.

APÊNDICE B – FORMULÁRIO DO SEGUNDO ESTUDO

Formulário - Más práticas de integração contínua na qualidade de software

IDENTIFICAÇÃO

E-mail:
Nível de escolaridade: ( )Graduação ( )Especialização ( )Mestrado ( )Doutorado ( )Pós-doutorado
Papel na organização: ( )Supervisor ( )Desenvolvedor ( )Testador ( )Design ( )Outro:_____
Nível de conhecimento em integração contínua: ( )Básico ( )Intermediário ( )Avançado
Nível de conhecimento em qualidade de software: ( )Básico ( )Intermediário ( )Avançado

PERGUNTA 01

As seguintes más práticas de CI podem impactar em quais indicadores de qualidade de software?

Má prática	Nenhum	Confiabilidade	Segurança	Manutenibilidade
Branches de funcionalidades são usadas em vez de toggles de funcionalidades				
Branches divergentes				
Desenvolvedores não têm controle completo do ambiente				
Notificações de falhas são enviadas apenas para equipes/desenvolvedores que se inscreveram explicitamente				
Testes ausentes em branches de funcionalidades				
Algumas tarefas do pipeline são iniciadas manualmente				

Falta de testes em um ambiente semelhante ao de produção				
Casos de teste não estão organizados em pastas com base em seus propósitos				
O hardware do servidor CI é usado para diferentes fins além de executar o framework CI				
Ferramentas externas são usadas com suas configurações padrão				
Um build falha devido a alguma instabilidade na execução, quando não deveria				
Scripts de build extensos				
Portões de qualidade são definidos sem os desenvolvedores considerarem apenas o que é ditado pelo cliente				
Desenvolvedores e operadores são mantidos como papéis separados				

PERGUNTA 02

Qual é o nível de esforço para resolver as seguintes más práticas de CI?

Má prática	Incerto	Baixo	Médio	Alto
Branches de funcionalidades são usadas em vez de toggles de funcionalidades				
Branches divergentes				
Desenvolvedores não têm controle completo do ambiente				
Notificações de falhas são enviadas apenas para equipes/desenvolvedores que se inscreveram explicitamente				
Testes ausentes em branches de funcionalidades				
Algumas tarefas do pipeline são iniciadas manualmente				
Falta de testes em um ambiente semelhante ao de produção				

Casos de teste não estão organizados em pastas com base em seus propósitos				
O hardware do servidor CI é usado para diferentes fins além de executar o framework CI				
Ferramentas externas são usadas com suas configurações padrão				
Um build falha devido a alguma instabilidade na execução, quando não deveria				
Scripts de build extensos				
Portões de qualidade são definidos sem os desenvolvedores considerarem apenas o que é ditado pelo cliente				
Desenvolvedores e operadores são mantidos como papéis separados				