UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE CIÊNCIAS EXATAS E DA TERRA
PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO
DOUTORADO ACADÊMICO EM CIÊNCIA DA COMPUTAÇÃO

The Effects of Continuous Integration on Software Development: A Causal Investigation

Eliezio Soares de Sousa Neto

Natal-RN, Brasil
2023

Eliezio Soares de Sousa Neto

The Effects of Continuous Integration on Software Development: A Causal Investigation

Tese de Doutorado apresentado ao Programa de Pós-Graduação em Sistemas e Computação do Centro de Ciências Exatas e da Terra da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do título de Doutor em Ciência da Computação. Área de Concentração: Engenharia de Software

Advisor: Uirá Kulesza
Advisor: Daniel Alencar da Costa

Natal-RN, Brasil
2023

Universidade Federal do Rio Grande do Norte - UFRN
Sistema de Bibliotecas – SISBI
Catalogação de Publicação na Fonte. UFRN - Biblioteca Setorial Prof. Ronaldo Xavier de Arruda - CCET

Sousa Neto, Eliezio Soares de.
The effects of continuous integration on software development: a causal investigation / Eliezio Soares de Sousa Neto . - 2023.
139 f.: il.

Orientação: Dr. Uirá Kulesza..
Coorientação: Dr. Daniel Alencar da Costa.
Tese (doutorado) - Universidade Federal do Rio Grande do Norte, Centro de Ciências Exatas e da Terra, Programa de Pós-Graduação em Sistemas e Computação. Natal, RN, 2023.

1. Computação - Tese. 2. Integração contínua - Tese. 3. Causalidade - Tese. 4. Qualidade de software - Tese. 5. Continuous integration - Tese. 6. Causation - Tese. 7. Software quality - Tese. I. Kulesza, Uirá. II. Costa, Daniel Alencar da. III. Título.

RN/UF/CCET CDU 004(043.2)

Elaborado por Joseneide Ferreira Dantas - CRB-15/324

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE CIÊNCIAS EXATAS E DA TERRA
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA
PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO

Ata nº. 118

ATA DA SESSÃO DE AVALIAÇÃO DE TESE DE DOUTORADO DO PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO.

Aos dezenove dias do mês de setembro de dois mil e vinte três (19/09/2023), às 17h30, por videoconferência, foi instalada a Comissão Examinadora responsável pela avaliação da tese de doutorado intitulada "***The Effects of Continuous Integration on Software Development: A Causal Investigation***", como trabalho final apresentado pelo(a) candidato(a) **ELIEZIO SOARES DE SOUSA NETO** ao Programa de Pós-Graduação em Sistemas e Computação, da Universidade Federal do Rio Grande do Norte, e parte dos requisitos para obtenção do título de **DOUTOR(A) EM CIÊNCIA DA COMPUTAÇÃO**. A Comissão Examinadora foi presidida pelo(a) professor(a) **Dr. UIRÁ KULESZA** (Orientador - UFRN) e contou com a participação de **Dr. DANIEL ALENCAR DA COSTA** (Orientador - University of Otago - NZL), **Dr. EDUARDO HENRIQUE DA SILVA ARANHA** (UFRN), **Dr. MARCELO DE ALMEIDA MAIA** (UFU), **Dr. RODRIGO BONIFACIO DE ALMEIDA** (UnB) e **Dr. SERGIO QUEIROZ DE MEDEIROS** (UFRN) na qualidade de examinadores. A sessão teve a duração de 4 horas e a Comissão emitiu o seguinte parecer: o trabalho e desempenho do candidato atenderam aos requisitos necessários a uma Tese de Doutorado, tendo a Comissão Examinadora, portanto **APROVADO** o trabalho.

Examinador(a) Externo(a): **Dr. DANIEL ALENCAR DA COSTA**

Documento assinado digitalmente
gov.br
EDUARDO HENRIQUE DA SILVA ARANHA
Data: 26/09/2023 10:17:51-0300
Verifique em https://validar.iti.gov.br

Examinador(a) Interno(a): **Dr. EDUARDO HENRIQUE DA SILVA ARANHA**

Documento assinado digitalmente
gov.br
MARCELO DE ALMEIDA MAIA
Data: 26/09/2023 11:41:49-0300
Verifique em https://validar.iti.gov.br

Examinador(a) Externo(a): **Dr. MARCELO DE ALMEIDA MAIA**

Documento assinado digitalmente
gov.br
RODRIGO BONIFACIO DE ALMEIDA
Data: 27/09/2023 11:04:25-0300
Verifique em https://validar.iti.gov.br

Examinador(a) Externo(a): **Dr. RODRIGO BONIFACIO DE ALM**

Documento assinado digitalmente
gov.br
SERGIO QUEIROZ DE MEDEIROS
Data: 25/09/2023 20:03:32-0300
Verifique em https://validar.iti.gov.br

Examinador(a) Interno(a): **Dr. SERGIO QUEIROZ DE MEDEIROS**

Documento assinado digitalmente
gov.br
UIRA KULESZA
Data: 25/09/2023 13:51:05-0300
Verifique em https://validar.iti.gov.br

Presidente: **Dr. UIRÁ KULESZA**

Documento assinado digitalmente
gov.br
ELIEZIO SOARES DE SOUSA NETO
Data: 25/09/2023 14:32:50-0300
Verifique em https://validar.iti.gov.br

Discente: **ELIEZIO SOARES DE SOUSA NETO**

Ao meu amado pai (in memoriam) que sempre viu e acreditou em um futuro que ainda não existia. Que sorria para sonhos que pareciam distantes demais. Meu pai, que inspirou e continua a inspirar a minha vida.

Agradecimentos

À Deus, criador de todas as coisas que, à sua semelhança, nos dotou de capacidade criativa, de curiosidade e de sonhos. A Ele dedico meu intelecto, meu esforço e todos os frutos, pois tudo é d'Ele, por Ele e para Ele. Que o título obtido com esta tese seja uma ferramenta a disposição d'Ele para abençoar vidas.

À minha amada esposa Jéssica e à minha preciosa filha Laura, que me apoiaram durante todo o processo, compartilhando as alegrias e desafios desta jornada. Abdicando do melhor do meu tempo em muitas situações, mas sempre vivendo o sonho desse doutorado junto comigo. Este trabalho também é de vocês. Obrigado.

Que a minha filha Laura encontre em mim e neste esforço uma memória valiosa de que tudo é possível com dedicação e disciplina. Lembre-se sempre que o por quê deve vir antes do como ou do quando. Viva por propósitos, entregue-se a Deus e será sempre bem sucedida.

À minha mãe Lucimar e meu pai Misael (in memoriam), que sempre me apoiaram incondicionalmente e cujo amor e incentivo continuam a inspirar minha jornada. Obrigado por tanta vida compartilhada e por sempre terem dado tudo de si a nós. Este trabalho está carregado de suas orações. Obrigado.

À minha irmã Mikaely, meus familiares e amigos, muito obrigado. Algumas vezes não estive com vocês, mas vocês nunca desistiram de mim. Este trabalho tem muito da fé, das orações e do apoio de vocês. Obrigado por celebrarem cada estudo, cada publicação, cada defesa. Obrigado.

Ao meu estimado orientador Uirá Kulesza que apostou em mim, abriu portas e com muita serenidade conduziu essa jornada pelas vias tortuosas da pesquisa acadêmica. Este trabalho e seus frutos se devem muito a você. Também a meu co-orientador Daniel Alencar da Costa que com seu apreço pela excelência me ajudou a extrair o melhor possível dos recursos que eu tinha. Obrigado por todo o tempo e paciência investidos na construção deste trabalho e na minha formação. Obrigado.

Ao Instituto Federal do Rio Grande do Norte, por fornecer os meios e oportunidades que tornaram possível a pesquisa e a escrita desta tese. Sou profundamente grato pela educação de qualidade que um dia recebi e por hoje exercer a docência nessa respeitosa e centenária instituição.

Muito obrigado a todos que tornaram este trabalho possível.

"A fé e a razão caminham juntas, mas a fé vai mais longe." — Agostinho de Hipona

Soli Deo Gloria.

Resumo

Integração Contínua (*Continuous Integration*—CI) é uma técnica de engenharia de software comumente mencionada como um dos pilares das metodologias ágeis. CI tem como principal objetivo reduzir o custo e o risco da integração de código entre times de desenvolvimento. Para tal se preconiza a realização de commits frequentes para integrar o trabalho dos desenvolvedores em um repositório de código e a frequente verificação de qualidade através de *builds* e testes automatizados. Através do uso de CI espera-se que os times de desenvolvimento possam detectar e corrigir erros rapidamente, melhorando a produtividade dos times e a qualidade dos produtos de software desenvolvidos entre outros benefícios apontados por pesquisadores e praticantes. Estudos anteriores sobre o uso de CI apontam diversos benefícios em diversos aspectos do desenvolvimento de software, entretanto tais associações não estão mapeadas como um todo e também não são suficientes para concluir que CI seja de fato a causa de tais resultados.

Portanto, este trabalho tem como objetivo investigar empiricamente tais efeitos da adoção de CI no desenvolvimento de *software* sob uma perspectiva causal. Primeiro, nós realizamos uma revisão sistemática de literatura para catalogar os achados de estudos que avaliaram empiricamente os efeitos da adoção de CI. Após explorar o conhecimento já documentado conduzimos dois estudos com o objetivo de aprofundar a compreensão a respeito de dois desses aspectos supostamente afetados pela adoção de CI: qualidade de software e a produtividade dos times de desenvolvimento. Nós pretendemos responder se há uma relação causal entre a adoção de CI e os efeitos reportados na literatura. Para isso utilizamos *causal Direct Acyclic Graphs (causal DAGs)* combinado a duas outras estratégias: revisão de literatura e um estudo de mineração de repositório de software (*Mining Software Repository*—MSR). Nossos resultados mostram um panorama dos efeitos de CI reportados na literatura e apontam que há de fato uma relação causal entre CI e qualidade de software.

Palavras-chave: Integração Contínua. Causalidade. Impacto. Engenharia de Software. Qualidade de Software. Produtividade.

Abstract

Continuous Integration (CI) is a software engineering technique usually mentioned as one of the foundations of agile methodologies. The main objective of CI is to reduce the cost and risk of code integration among development teams. For such, it preconizes frequent commits to integrate the work from developers into a source code repository and the frequent quality verification via automated builds and tests. Through CI usage, it is expected that development teams can quickly detect and correct issues, improving team productivity and software quality, among other benefits pointed out by researchers and practitioners. Previous studies regarding CI usage highlight several benefits in software development aspects. However, such associations are not mapped as a whole and are not sufficient to conclude that CI is indeed the cause of such results.

Therefore, the main goal of this work is to investigate the effects of CI adoption on software development from a causal perspective. First, we conducted a systematic literature review to catalog the findings from studies that empirically evaluated the effects of adopting CI. After exploring the existing state-of-the-art, we conducted two studies to deepen the comprehension regarding two aspects supposedly impacted by CI: software quality and teams' productivity. We investigate if there is a causal relationship between CI adoption and such literature-reported effects. For this purpose, we employ causal Direct Acyclic Graphs (causal DAGs) combined with two other strategies: a literature review and a mining software repository (MSR) study. Our results show a panoramic view of CI literature-reported effects and point out that, indeed, there is a causal relationship between CI and software quality.

Keywords: Continuous integration. Causation. Impact. Software Engineering. Software Quality. Productivity.

List of Figures

List of Tables

List of abbreviations and acronyms

CD	Continuous Deployment
CDE	Continuous Delivery
CI	Continuous Integration
DAG	Directed Acyclic Graph
ESE	Empirical Software Engineering
IMGD	Integrated Model of Group Development
ISO	International Organization for Standardization
MWW	Mann-Whitney-Wilcoxon test
MSR	Mining Software Repository
OSS	Open-Source Software
PR	Pull Request
RDD	Regression Discontinuity Design
SLR	Systematic Literature Review
TDD	Test-Driven Development
UFRN	Universidade Federal do Rio Grande do Norte
XP	eXtreme Programming

List of symbols

$\rightarrow$	Causation
$\not\perp\!\!\!\perp$	Not independent
$\perp\!\!\!\perp$	Independent
$\mid$	Conditioned on
$\Rightarrow$	Implies

Contents

1 Introduction

Continuous integration (CI) is a software engineering practice that aims to reduce the costs and risks related to code integration among distributed teams through frequent code integration and synergic practices, such as automated tests, frequent builds, and immediately fixing a broken build, among others (BECK; ANDRES, 2004; FOWLER; FOEMMEL, 2006; DUVALL; MATYAS; GLOVER, 2007; STåHL; BOSCH, 2014a). CI has increased popularity in a broad range of domains in the recent decades (BECK; ANDRES, 2004; FOWLER; FOEMMEL, 2006; DUVALL; MATYAS; GLOVER, 2007; STåHL; BOSCH, 2014a).

The literature points out several potential benefits related to CI, such as risk reduction, greater confidence in the software product, ease of locating and fixing bugs, improvements in project predictability, team communication, software quality, and gains in team productivity, among others (FOWLER; FOEMMEL, 2006; DUVALL, 2013; STåHL; BOSCH, 2013; VASILESCU et al., 2015; SOARES et al., 2022). In an increasingly globalized scenario and distributed software development teams, the potential of CI attracts even more attention from the industry since the distributed character demands strong coordination and control from software development teams facing temporal, geographical, and socio-cultural challenges (KAUSAR; AL-YASIRI, 2015; PEHMÖLLER; SALGER; WAGNER, 2021; PHALNIKAR; DESHPANDE; JOSHI, 2009; HOLMSTROM et al., 2006).

The CI popularity has also caught the attention of the software engineering research community. Several studies investigated CI practices (VASSALLO; PALOMBA; GALL, 2018; YU et al., 2016; PINTO et al., 2018), and associated environments & tools (STAHL; BOSCH, 2014; ZAMPETTI et al., 2017; JOHANSSEN et al., 2018). Other studies have focused on the potential benefits of CI on the delivery time of pull requests (BERNARDO; COSTA; KULESZA, 2018), on build health (EMBURY; PAGE, 2019), and in software development (STåHL; BOSCH, 2013). There are also studies exploring challenges (DEBBICHE; DIENéR; SVENSSON, 2014a), long builds (GHALEB; COSTA; ZOU, 2019), build failures (RAUSCH et al., 2017), and anti-patterns (VASSALLO et al., 2019) linked to CI usage. Some studies investigated new practices introduced to CI (ROGERS, 2004; VOLF; SHMUELI, 2017; MEEDENIYA; RUBASINGHE; PERERA, 2019) in different project settings.

1.1 Problem Statement

Over the last years, the software engineering research community has produced many studies related to the CI impact. Among the vast literature, Michael Hilton and colleagues presented two studies with an association between CI and high-quality code and tests,

leading to less time to identify and reject problematic pull requests (HILTON et al., 2016; HILTON et al., 2017). Several studies related CI to earlier bug catching and an increased number of issues and bugs resolved (KAYNAK; ÇILDEN; AYDIN, 2019; RAHMAN et al., 2018; PINTO et al., 2018). Bogdan Vasilescu et al. showed that CI improves teams' productivity without compromising code quality (VASILESCU et al., 2015). Similarly, Jadson Santos et al. presented an association of CI sub-practices with software quality and teams' productivity (SANTOS; COSTA; KULESZA, 2022). In addition to these, other studies also demonstrate the association with teams' productivity (STåHL; BOSCH, 2013; PARSONS; RYU; LAL, 2007). Although correlation studies are valuable, causal studies can provide deeper insights and empower stakeholders to make better decisions, such as adopting or not CI in their teams (PEARL et al., 2000).

This thesis explores the existing literature limitations, especially the lack of causal studies on the effects of CI on software development. To understand how CI influences the delivery of software products, this thesis investigates the potential causal relationship between CI and software quality and CI.

1.2 Thesis Proposal

This thesis aims to identify the reported claims regarding the effects of CI on software development and to propose a deeper investigation into some of these claims, approaching them from a causal perspective. To accomplish this goal, we propose three studies as shown in Figure 1.

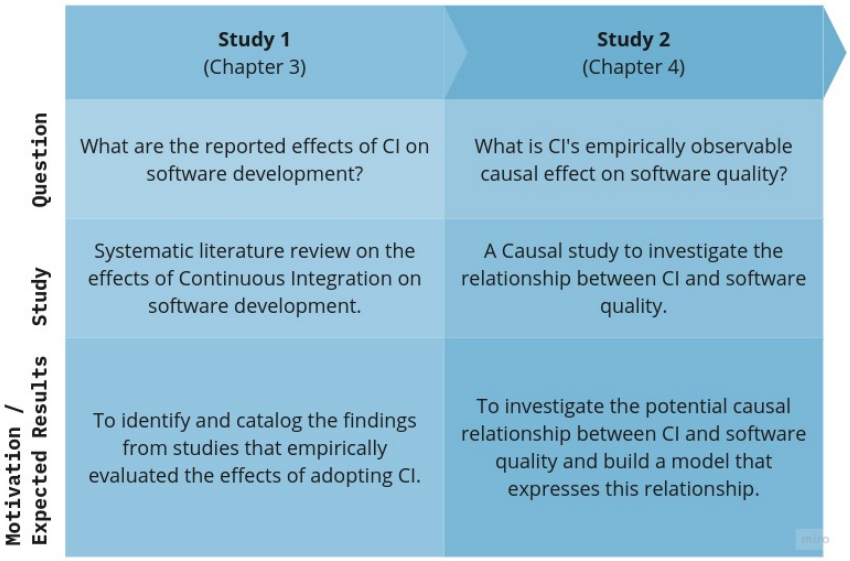


Figure 1 – Thesis overview

Study 1 (1.2.1) explores the current literature on CI and maps the reported effects of CI adoption on software development. In subsequent studies, we investigate the mapped

relationships of CI with the reported effects from a causal perspective. In Study 2 (1.2.2), we investigate the potential causal relationship between CI and software quality.

1.2.1 Study 1: What are the reported effects of CI on software development?

Current research has cataloged the findings in the literature concerning continuous integration, delivery, and deployment (LAUKKANEN; ITKONEN; LASSENIUS, 2017; SHAHIN; BABAR; ZHU, 2017; STåHL; BOSCH, 2013). However, no systematic study summarizes all the potential benefits and cons of using CI (i.e., the effects of adopting CI on the development process). This kind of study could better inform practitioners and researchers about the potential of using CI and future research avenues. We aim to investigate how the existing research evaluated CI and its results. Additionally, we also intend to investigate the criteria used to identify whether a given project uses CI or not—which is essential for designing empirical studies related to CI—, and what the research methodologies applied in the existing studies to evaluate the potential effects of adopting CI.

Our work is a *systematic literature review* (SLR) (KEELE et al., 2007) of the existing empirical evidence regarding the effects of CI in diverse software development activities. In this way, we consider a diverse set of empirical methods and associations with CI, i.e., the effects of CI on variables such as test coverage, bugs reported, and team communication, among others. Assuming this variability in the empirical methods and diversity of variables, we do not perform a meta-analysis. Instead, we present an interpretive SLR to draw a picture of the reported benefits and cons of adopting CI and collate the claims made about CI in the existing literature, systematically assessing these claims' strengths. In this way, this work offers meaningful and relevant evidence-based support for practitioners, organizations, and researchers.

We discuss the findings regarding the effects of CI and their evidence across six themes: *development activities*, *software process*, *quality assurance*, *integration patterns*, *issues & defects*, and *build patterns*. These findings provide researchers and practitioners with (i) state-of-the-art empirical claims related to the effects of CI while collating their existing evidence; and (ii) insights regarding the interrelation between research methodologies, quality assessment and themes, which delineate potential future studies.

The results of this study are essential for knowing the effects associated with CI, and then we can investigate them under a causal perspective in the following studies.

1.2.2 Study 2: What is CI's empirically observable causal effect on software quality?

Study 1 revealed that the existing literature reported several benefits associated with CI usage. Among other associations, CI is associated with improvements in testing practices, better quality assurance, and reduced issue reports, among others (SOARES et al., 2022). This study aims to investigate the potential causal relationship between CI and software quality to understand the CI influence on delivering software products.

To study the causal relationship between CI and software quality, we use an approach that consists of five interconnected stages. In Stage 1, we conduct a literature review to understand the variables that can play a role in the relationship between CI and software quality, as well as marginal associations. With these variables, in Stage 2, we define a comprehensive causal DAG (i.e., a graphical-statistical technique - to enable us to draw domain assumptions and infer causal conclusions in a later stage (HERNáN; ROBINS, 2010)).

Having a causal DAG containing a sufficient set of variables to analyze the relationship between *ContinuousIntegration* and *BugReport* (we consider bug reports a proxy for software quality, similar to a previous study from Vasilescu et al. (VASILESCU et al., 2015)), we can proceed with Stage 3. This stage consists of mining software repositories to collect observational data on the variables of the DAG, allowing us to apply the d-Separation rules (PEARL; JUDEA, 1994) and evaluate the raised set of testable statistical implications from the DAG built in Stage 1 (d-Separation is explained in Section 2.4.1).

Stage 4 verifies the statistical implications of the causal DAG by performing (un)conditional independence tests on our dataset. Finally, in Stage 5, we analyze the hypotheses that failed in Stage 4 (i.e., statistical implications from the supposed relationships between the variables not supported by the data) and propose a new causal DAG. The DAG expresses the causal paths between CI and bug reports and which other variables play relevant causal influences.

1.3 Thesis Contributions

This thesis presents a set of contributions through two studies. In study 1, we present findings about the benefits of CI usage. CI is mentioned as a success factor in software projects (Section 3.2.2.2), improving productivity, efficiency, and developer confidence (Sections 3.2.2.1 and 3.3.2.2). CI promotes benefits in the development process (Section 3.2.2.2) and potentializes pull-based development by improving and accelerating the integration process (Section 3.2.2.4).

Study 1 also shows indications that CI positively influences the way developers work (Section 3.2.2.4) and demonstrates a perceived provision of transparency and continuous

quality inspections (Section 3.2.2.3). In addition, the studies credit CI to an improvement in the time to find and fix issues and a decrease in defects reported (Section 3.2.2.5).

On the other hand, Study 1 shows drawbacks of CI usage, such as the introduction of complexity to the project, requiring more effort and discipline from developers, and negatively impacting developers' perceived productivity (Sections 3.2.2.1 and 3.3.2.2). Some of the included studies also discuss the false sense of confidence, i.e., when developers blindly rely on flaky tests (Sections 3.2.2.1 and 3.3.2.2). Studies also report that CI may prolong the pull request lifetime (Section 3.2.2.4).

The study 2 assess empirically the causal effect of CI on software quality. We built a literature-based causal DAG expressing such a relationship (Section 4.2.1) and mined software repositories to assess the literature-based causal DAG empirically, analyzing 12 activity months from 148 software projects (Section 4.2.2). After analyzing the testable implications from the literature-based causal DAG with our dataset, we proposed a new data-validated causal DAG (based on a hybrid literature-data approach) expressing the relationship between CI, software quality, and the relevant variables (Section 4.2.3).

With the causal analysis, we find that CI has a positive causal effect on bug reports and influences developers' communication, reinforcing bug report benefits. On the other hand, the developers' overconfidence is a concern in CI environments and could negatively affect bug reports. CI also has an impact on Merge Conflicts and commit frequency.

It is essential to notice that commit frequency, test volume and communication are interrelated, and all of them affect Bug Report. In the same way, Age is an important source of confounding effects since it relates to test volume, commit frequency, continuous integration, communication, and bug report.

1.4 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 describes essential concepts covered in the thesis scope. Chapter 3 presents our systematic literature review (Study 1) and their findings. Chapter 4 presents the causal explanatory study on CI and software quality (Study 2). Chapter 5 situates the thesis concerning other related studies. Finally, Chapter 6 draws the conclusions, implications, and future work.

2 Background

This chapter provides concepts, definitions, techniques, and approaches to understand this thesis better. We present definitions of Continuous Integration (CI) in Section 2.1, regarding software quality in Section 2.2, and background material regarding causal DAGs in Section 2.4.

2.1 Continuous Integration

Continuous Integration (CI) is one of the practices of eXtreme Programming (XP) methodology proposed by Beck K (BECK; ANDRES, 2004). The overarching goal of CI is to reduce the cost of integrating the code developed by different developers in a team (or different teams) by making integration a daily practice. For example, there must be no more than a couple of hours between code integration. While CI compels the code to be collective and the knowledge to be shared more, CI's main benefit is the reduced risk of a big and cumbersome integration (e.g., after days, weeks, or months of work developed) (BECK; ANDRES, 2004).

To properly employ CI, at least four mechanisms are required: (i) a version control system, (ii) a build script, (iii) a feedback mechanism, and (iv) a process for integrating the source code changes (DUVALL, 2013). Modern distributed version control systems (VCS), especially those based on Git, have grown in popularity because of social coding platforms, such as GitHub (VASILESCU et al., 2015), which have fostered collaborative software development. Within these popular social coding platforms, several services have been proposed to support CI (e.g., TravisCI, CircleCI and Jenkins), easing the automation of build pipelines, which are triggered by source code changes on the VCS (HILTON et al., 2016).

Studies have reported an increasing number of projects adopting the continuous integration practice (HILTON et al., 2016), and some of such studies bring up evidence showing changes in the practice of these projects, such as higher commit frequency and an increase in test automation (ZHAO et al., 2017).

Duvall et al. (DUVALL, 2013) advocate that CI is the centerpiece of software development, ensuring the health and determining software quality. To get the benefits of CI, the authors argue that developers should implement a set of sub-practices daily, whereas implementing only a fraction of the practices is not enough to employ CI. Fowler, in his definition of CI, also mentions a series of critical practices to make CI effective (FOWLER; FOEMMEL, 2006) (see Table 1).

Nevertheless, some authors have studied differences in implemented CI processes and demonstrated a lack of consensus regarding these CI processes, which results in many

CI variants (STåHL; BOSCH, 2014b; VIGGIATO et al., 2019). Ståhl & Bosch (STåHL; BOSCH, 2014b) identified variation points from 16 out of 22 clusters of CI practices and argue that it is necessary to investigate which kind of continuous integration a project applies when analyzing or assessing projects. Viggiato et al. (VIGGIATO et al., 2019) suggested that continuous integration may not always be homogeneous, i.e., CI may have different usages across different domains. Studies still suggest the inclusion of other practices to potentialize benefits, Vassalo C et al. (VASSALLO; PALOMBA; GALL, 2018), for example, suggest adding "*continuous refactoring*" as a CI best practice as it is useful to control the increasing complexity of the changes.

On top of that, there is a discussion regarding existing confusion around the definition of Continuous Integration (CI), Continuous Delivery (CDE), and Continuous Deployment (CD), or still, the recent emphasis on DevOps shedding light on the integration between software development and its operational deployment (SHAHIN; BABAR; ZHU, 2017; FITZGERALD; STOL, 2014). A conservative perspective presents these continuous practices as sequential and well-defined techniques, i.e., CI as a foundation for CDE in such a manner that an organization should implement a reliable CI practice to adopt CDE, in the same way, to implement CD an organization should implement CDE (SHAHIN; BABAR; ZHU, 2017). Fitzgerald B & Stol K (FITZGERALD; STOL, 2014), in turn, defends a holistic view —"Continuous ∗"— including Business Strategy & Planning, Development, and Operations, in which CI incorporates CDE, and CD.

Therefore, the variability around the continuous practices and the dynamic nature of the employed practices in continuous integration leads to a potentially endless variation of CI implementations and a lack of consensus on an exact definition of CI. Considering the lack of consensus regarding an exact definition of CI, in our research, we focus on the practices discussed by Duvall et al. (DUVALL, 2013) and Fowler (FOWLER; FOEMMEL, 2006)for one main reason. While other authors reveal the variability around continuous integration, they often do not provide concrete guidelines as to what should be considered CI or not. Conversely, Duvall et al. (DUVALL, 2013) and Fowler (FOWLER; FOEMMEL, 2006) present a concrete minimum number of practices that projects should adopt in order to be considered as using CI.

2.1.1 Continuous Integration Practices

Table 1 shows an overview of the practices proposed by Duvall et al. (DUVALL, 2013) and those reported by Fowler (FOWLER; FOEMMEL, 2006). The practices proposed by Duvall are shown in the second column, while the third column shows the practices reported by Fowler. In the first column, we organize the CI practices into four groups: (i) integration, (ii) test, (iii) build, and (iv) feedback.

"*Commit code frequently*" is the practice of integrating code changes as "*early and often*" as possible to a "*single source code repository*" (e.g., GitHub, GitLab, or

Table 1 – Continuous integration practices enumerated by Duvall et al. (DUVALL, 2013) and Fowler (FOWLER; FOEMMEL, 2006).

	Duvall et al. practices (DUVALL, 2013)	**Fowler practices (FOWLER; FOEMMEL, 2006)**
Integration Practices	Commit code frequently	Everyone commits to the mainline every day
	-	Maintain a single source repository
Test Practices	Write automated developer tests	Make your build self-testing
	All tests and inspections must pass	Test in a clone of the production environment
	-	Make it easy for anyone to get the latest executable
	-	Automate deployment
Build Practices	Don't commit broken code	Automate the build
	Run private builds	Every commit should build the mainline on an integration machine
	Fix broken builds immediately	Fix broken builds immediately
	-	Keep the build fast
Feedback Practices	Avoid getting broken code	Everyone can see what's happening

Bitbucket). This practice is central to CI because it prevents a complex integration—an integration that requires more time and effort—while treating potential integration problems (DUVALL, 2013; FOWLER; FOEMMEL, 2006).

When it comes to testing, CI bears the principle that "*all tests and inspections must pass*". This practice advocates that not only tests must pass but also the inspections related to coding and design standards (e.g., test coverage, cyclomatic complexity, or others). Ideally, the tests and inspections should be triggered in an automated fashion. Therefore, CI requires developers to "*write automated development tests*", "*making the builds to become self-testing*", which enables a fully automated build process that provides meaningful feedback (DUVALL, 2013; FOWLER; FOEMMEL, 2006).

Still regarding tests, Fowler recommends to "*test the software in a clone of the production environment*" to mitigate the risk of not identifying problems occurring only within the production environment. For this reason, Fowler also proposes the "*automated deployment*"—to prepare test-environments automatically—and the practice of "*making it easy for anyone to get the latest executable*"—so that anyone has easy access to the current state of development (FOWLER; FOEMMEL, 2006).

Concerning building practices, the team must follow the "*don't commit broken code*" practice. To do so, it is vital to employ the "*automate the build*" practice. The build automation consists of empowering the team with scripts that fully manage the build process, from dependency managers (e.g., Maven, Gradle, NuGet, or Bundler) and tests to a database schema, or other required tool. Once a consistent build script is set, developers should "*run private builds*" that emulate an integration build in their workstation, ensuring a well-succeeded build process before integrating their changes into the central repository (i.e., the mainline) (DUVALL, 2013; FOWLER; FOEMMEL, 2006).

Additionally, Fowler recommends that "*every commit should build the mainline on an integration machine*", i.e., a change sent to the mainline repository must trigger a build

process in a dedicated server. It is also important to "*keep the build fast*", so the dedicated server can be effective to give rapid feedback, helping developers to "*fix broken builds immediately*". Regarding build duration, the eXtreme Programming (XP) recommends a limit of 10 minutes. Builds that take more than 10 minutes may lead the development team to give up on using CI (FOWLER; FOEMMEL, 2006; BECK; ANDRES, 2004).

"*Fix broken builds immediately*" is cited both by Fowler and Duvall et al. A build may break due to a compilation error, a failed test, or several other reasons. When a build is broken, the development team must focus on fixing the build before any other implementation activity—the build should always be *on green*.

There are also CI practices related to feedback. One example is the practice "*everyone can see what's happening*", which makes communication clear and transparent within or across development teams. The immediate feedback from CI allows the development team to "*avoid getting broken code*". In other words, a developer can check the current build status before performing a checkout (or pull) (FOWLER; FOEMMEL, 2006; BECK; ANDRES, 2004).

2.2 Software Quality

Quality is a complex concept that may represent a subjective or a concrete concern. Garvin, David A (GARVIN, 1984) points out several approaches to answer what quality is:

> Five major approaches to the definition of quality can be identified: (1) the transcendent approach of philosophy; (2) the product-based approach of economics; (3) the user-based approach of economics, marketing, and operations management; and (4) the manufacturing-based and (5) value-based approaches of operations management (GARVIN, 1984).

Quality is frequently defined as "conformance to requirements" (HOYER; Y., 2001). Not so far, the software development industry traditionally defined software quality as "fit for purpose" or "conforming to specification" (BARNEY et al., 2012). In this sense, a lot of research on software quality aims to improve individual aspects of software quality, such as maintainability, security, or usability, while there are several models of software quality to support the software development process (BARNEY et al., 2012).

According to ISO/IEC 25010:2011 (International Organization for Standardization, 2011) the characteristics defined by the quality models are relevant and "provide consistent terminology for specifying, measuring, and evaluating system and software product quality." As stated by Ian Sommerville (SOMMERVILLE et al., 2011), measuring some quality attributes directly is impossible since their specification cannot be non-ambiguous and is challenging to measure. Measurement concerns the derivation of a numeric value or a profile for an attribute of a software component, system, or process, with which we can evaluate the software methods, tools, and processes (SOMMERVILLE et al., 2011).

Previous research works investigate quality issues using the number of bug reports as a metric for quality assessment (KHOMH et al., 2012; VASILESCU et al., 2015). In this work, we adopt a similar strategy considering bug reports as a proxy for software quality in study 2 (Chapter 4).

2.3 Common Cause Principle

Reichenbach's Common Cause Principle (PENROSE; PERCIVAL, 1962) states that given two statistically dependent variables X and Y, if one is not a cause of the other, then they may share a common cause Z, as shown in Fig. 2(a). Conditioning on the common cause Z, then X and Y become independent. For example, Fig. 2(a) shows an association "flowing" (the red dots above on the edges represent an "association flow") between $Heat$ and $Smoke$. They are associated because they share a common cause, which is $Fire$. Fig. 2(b) shows the interrupted flow when conditioning on $Fire$, i.e., in the absence of $Fire$ (i.e., $Fire$=0), there is no association between $Heat$ and $Smoke$.

Therefore, if the famous adage states that "correlation does not imply causation" (i.e., statistical associations are not sufficient to determine causal relationships), on the other hand, "there is no causation without association." The common cause principle establishes a relationship between statistical properties (i.e., association) and causal structures (PETERS; JANZING; SCHöLKOPF, 2017). In this way, it is possible to infer the existence of causal links from statistical dependencies (i.e., functional relationships between the variables) (PETERS; JANZING; SCHöLKOPF, 2017). To infer causation, Pearl (PEARL et al., 2000) proposed employing a causal modeling framework based on causal Directed Acyclic Graphs (causal DAGs). In the following subsections, we explain the theory proposed by Pearl because we apply his proposed theory in our study.

2.4 Causal Directed Acyclic Graphs (Causal DAGs)

Pearl (PEARL et al., 2000) argues that nature possesses causal mechanisms that, if described in detail, are deterministic functional relationships between variables. Some of these variables are unobservable, e.g., sometimes we see the smoke causing a fire alarm to be activated, but we can not see the fire. However, it is the fire that causes the existence of smoke. Pearl describes the causal discovery task as an induction game that contributes to identifying (from available observations or interventions) the organization of the mechanisms in the form of an acyclic causal structure (PEARL et al., 2000). This causal structure is called a Directed Acyclic Graph (DAG). A DAG has: (i) nodes - that are variables with directed edges and no directed cycles; and (ii) edges that represent functional relationships between variables (see Fig. 2(a)) (PEARL et al., 2000; SPIRTES; GLYMOUR; SCHEINES, 1993; HERNáN; ROBINS, 2010).

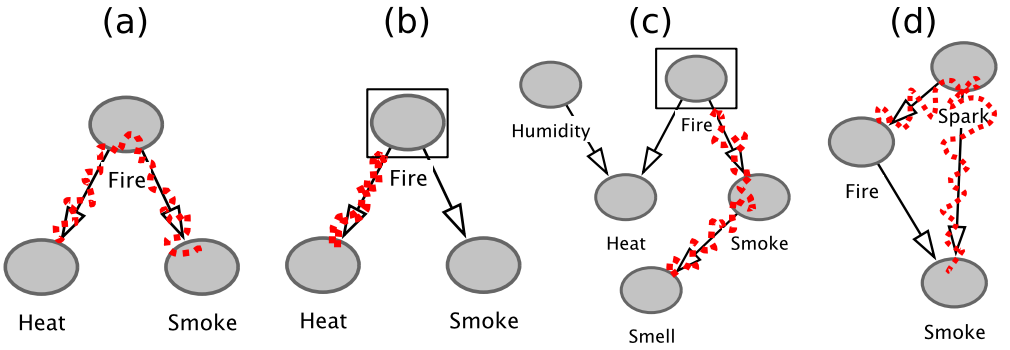


Figure 2 – In this picture, the ellipses are variables, the edges represent a relationship between variables, and the red dots above the edges represent an "association flowing" between variables. (a) Fire is the common cause of Heat and Smoke. Heat and Smoke are associated through Fire, i.e., the "association flows" between Heat and Smoke through Fire. (b) Conditioning on Fire, the association flow between Heat and Smoke is blocked. (c) Conditioning on Fire, Smoke becomes associated only with its descendent Smell. (d) Spark is a common cause of Fire and Smoke, opening a *backdoor path* between these variables. Spark is a source of confounding.

In turn, Pearl and Verma (PEARL; VERMA, 1995) defined a Causal Markov Condition stating that a DAG should have a node distribution $N = \{N_1, ..., N_n\}$ such that, for each j, N_j is independent of its non-descendants conditioning on its parents (PEARL et al., 2000; HERNáN; ROBINS, 2010; SPIRTES; GLYMOUR; SCHEINES, 1993). Considering the example of Fig. 2(c), the Markov Condition implies that, if conditioning on the parent $Fire$, $Smoke$ becomes independent of all other variables on the DAG, except its descendent $Smell$. That means that $Smoke$ exists ($Smoke = 1$), but there is not necessarilly $Heat$, because we condition on $Fire$ ($Fire = 0$).

Reichenbach's Common Cause Principle (see section 2.3) and the Markov Condition imply that a causal DAG should contain the common causes of any pair of variables (PEARL et al., 2000; HERNáN; ROBINS, 2010). Thus, to build a sufficient causal DAG of a phenomenon, it is essential to know the significant common causes among the variables involved in that phenomenon, i.e., if two variables $Heat$ and $Smoke$ in the DAG share a common cause $Fire$, then $Fire$ should be represented in the DAG, as shown in Fig. 2(c). Since this is a recursive criterion, in Fig. 2(d), $Spark$ should be present since it is a common cause of $Fire$ and $Smoke$.

Discovering causal structures to build a DAG that correctly describes a phenomenon, i.e., the set of variables a DAG should contain and the relationships between variables, is challenging. There are at least three strategies to obtain a DAG: prior knowledge, guessing-and-testing, or discovery algorithms (SHALIZI, 2021). Prior knowledge is a source to build causal DAGs, but since there is a link between causal structures and statistical properties (see section 2.3), it is possible to test the correctness of a built DAG if we have access to observational data. Section 2.4.1 details the statistical properties and

their testable implications.

2.4.1 d-Separation and the Testable Implications of the DAGs

A helpful approach to visually understand DAGs is to assume that associations "flow" through the edges of the DAG (HERNáN; ROBINS, 2010). Using the illustration from Fig. 2(a), the association flows freely between $Heat$ and $Smoke$. However, intervening on the values of $Fire$ (as in the Fig. 2(b)), this flow may be interrupted since it represents a common cause for $Heat$ and $Smoke$ (HERNáN; ROBINS, 2010). A set of graphical rules, called d-separation, were formalized by Pearl (PEARL; JUDEA, 1994) to infer associational conclusions from causal DAGs.

D-separation is a set of graphical rules defining whether a path in the DAG is blocked or open. To understand the rules, we consider three structural patterns: (i) Chain: $Fire \rightarrow Smoke \rightarrow Smell$ (see Fig. 2(c)); (ii) Fork: $Heat \leftarrow Fire \rightarrow Smoke$ (see Fig. 2(a)); and (iii) Collider: $Humidity \rightarrow Heat \leftarrow Fire$ (see Fig. 2(c)). In the chain, $Fire$ and $Smell$ are dependent ($Fire \not\perp\!\!\!\perp Smell$) but become independent (i.e., blocked), conditioning on $Smoke$ ($Fire \perp\!\!\!\perp Smell \mid Smoke$). In the fork, $Heat$ and $Smoke$ are dependent ($Heat \not\perp\!\!\!\perp Smoke$) unless we condition on $Fire$ ($Heat \perp\!\!\!\perp Smoke \mid Fire$). In the collider, $Humidity$ and $Fire$ are independent ($Humidity \perp\!\!\!\perp Fire$) because a collider blocks the association flow. Conditioning on the collider (or one of its descendants) opens that flow, making $Humidity$ and $Fire$ dependent(e.g., $Humidity \not\perp\!\!\!\perp Fire \mid Heat$). We provide a summary below:

- Chain: $X \rightarrow Z \rightarrow Y \Rightarrow X \not\perp\!\!\!\perp Y$ and $X \perp\!\!\!\perp Y \mid Z$
- Fork: $X \leftarrow Z \rightarrow Y \Rightarrow X \not\perp\!\!\!\perp Y$ and $X \perp\!\!\!\perp Y \mid Z$
- Collider: $X \rightarrow Y \leftarrow Z \Rightarrow X \perp\!\!\!\perp Z$ and $X \not\perp\!\!\!\perp Z \mid Y$

Based on these d-Separation rules, all DAGs have a consequent set of testable statistical implications. These statistical implications inferred from the graphical analysis allow us to verify if the structure of a DAG is consistent with an empirical dataset through conditional independence tests on the data. Pearl (PEARL et al., 2000) defines a causal model as a pair $M =< D, \Theta_D >$ consisting of a causal structure D and a set of parameters Θ_D compatible with D. Thus, relying on d-Separation rules and a representative dataset, the causal DAG technique also allows building a causal structure through both guessing-and-testing as well as discovery algorithms (SHALIZI, 2021).

This study aims to understand how Continuous Integration affects Software Quality. We use a combination of two approaches: (i) prior knowledge to draw the existing assumptions; and (ii) guessing-and-testing to obtain a final causal structure. To introduce more rigor, we consider prior knowledge by gathering it from the existing literature and

drawing an initial DAG with assumptions regarding the influence of CI on software quality. Afterward, we refine our understanding by applying guessing-and-testing that relies on the statistical implications that we derive from the initial DAG and their validations by using our dataset, i.e., we collect data by mining software repositories. Section 4.1 details these methodological steps.

2.4.2 Backdoor Paths and Confounding

A common source of bias is due to common causes between two variables, such as the presence of a cause *Spark* shared by the treatment $Fire$ and the outcome *Smoke* (see Fig. 2(d)). This common cause results in a path between $Fire$ and *Smoke* that is not a direct edge. The path is through the back door, i.e., $Fire \leftarrow Spark \rightarrow Smoke$ generating another association flow between the treatment $Fire$ and the outcome *Smoke*. We refer to this kind of structure as a *backdoor* path, and the bias caused by the backdoor path as *confounding*.

Causal DAGs have grown in popularity in several fields (econometrics, epidemiology, and climate science, among others). Greenland, Pearl, and Robins (GREENLAND; PEARL; ROBINS, 1999a) present causal DAGs as a tool for identifying variables that must be measured and controlled to obtain unconfounded causal effect estimates for epidemiologic research. Shmueli (SHMUELI, 2010) sees causal DAGs as a common causal inference method for testing causal hypotheses on observational data.

3 Systematic Literature Review on The Effects of Continuous Integration on Software Development

An earlier version of this chapter appears in Volume 27, issue 3, from May 2022 of Empirical Software Engineering (SOARES et al., 2022).

3.1 Research Method

The main goal of our study is to provide a holistic view for researchers and practitioners regarding how Continuous Integration (CI) can influence the software development phenomena (both in terms of potential benefits and cons). Therefore, we conduct a *Systematic Literature Review* (SLR) of studies that investigated the potential effects of CI on software development. To evaluate the scientific rigor of our target studies, we investigate the methodologies that were employed in these studies. The purpose of this investigation is better to understand the strength of the existing scientific claims and inform the reader accordingly. We also consider how our target studies determined whether their subject projects used CI or not. Identifying whether a project uses CI is a crucial step in any study evaluating the effects of adopting CI as this is how empirical comparisons regarding CI vs. non-CI can be performed. To conduct our SLR, we follow the guidelines provided by Kitchenham & Charters (KEELE et al., 2007).

The next subsections describe our review protocol (KEELE et al., 2007). Section 3.1.1 describes the rationale behind our research questions. Section 3.1.2 details the search mechanisms that we perform. Section 3.1.3 describes the inclusion and exclusion criteria and the screening process. Section 3.1.4 describes the data extraction details, while Section 3.1.5 explains how we assess the quality of the studies. Finally, Section 3.1.6 reveals the procedures that we use to synthesize the collected data.

3.1.1 Research questions

To fulfill the goal of our study, we address the following research questions (RQs):

RQ1: What are the existing criteria to identify whether a software project uses CI?

Rationale. Several authors have listed a set of practices or principles related to CI (DUVALL, 2013; FOWLER, 2020; STåHL; BOSCH, 2014b; ZHAO et al., 2017;

VIGGIATO et al., 2019). Some of these practices include: "commit code frequently", "test automation", "run private builds", "all tests and inspections must pass", and "fix broken builds immediately". However, evidence exists that many CI projects do not adopt many of these practices.

For example, Felidré et al. (FELIDRé et al., 2019) analyzed 1,270 open-source projects using TravisCI (the most used CI server). They observed that about 60% of the projects do not follow proper CI practices. For example, some projects have infrequent commits, low test coverage, and 85% of projects take more than four days to fix certain builds. Therefore, in RQ1, we investigate which criteria have been applied in the studies to identify whether the subject projects employ CI or not. This investigation is important because it has a direct impact on the quality of the data. For example, if a project is deemed to be using CI, but performs infrequent commits and takes a long time to fix builds, the empirical results observed to such a project would not reflect proper CI usage.

RQ2: What are the reported claims regarding the effects of CI on software development?

Rationale. Most practitioners adopt CI practices with the expectation of increasing the quality of software development (LEPPäNEN et al., 2015). Researchers have reported the benefits of applying CI (FOWLER; FOEMMEL, 2006; DUVALL, 2013), such as risk reduction, decrease in repetitive manual processes, readily deployable software, improved project visibility, greater confidence in the software product, and easiness of locating and removing defects.

To help practitioners and researchers, from an evidence-based software engineering effort (KITCHENHAM; DYBå; JøRGENSEN, 2004), this RQ aims to collect, organize, and compare the empirical investigations related to CI performed by existing studies, while highlighting the assumptions and claims associated with these empirical investigations.

RQ3: Which empirical methods, projects and artifacts are used in the studies that investigate the effects of CI on software development?

Rationale. As observed by Easterbrook S et al. (EASTERBROOK et al., 2008), there is a lack of guidance regarding which methods to apply in *Empirical Software Engineering* (ESE) studies—which leads many researchers to select an inappropriate methodology. Rodríguez-Pérez et al. (RODRIGUEZ-PéREZ; ROBLES; GONZáLEZ-BARAHONA, 2018) investigated the reproducibility aspects of ESE through a case study. According to their investigations, 39% of the analyzed papers did not provide sufficient data or documentation to support the reproduction of the studies. To better understand the methodologies applied in the ESE field concerning CI, in this RQ, we shed light on the methods, evaluations, domains, and kinds of projects that are investigated in our target studies.

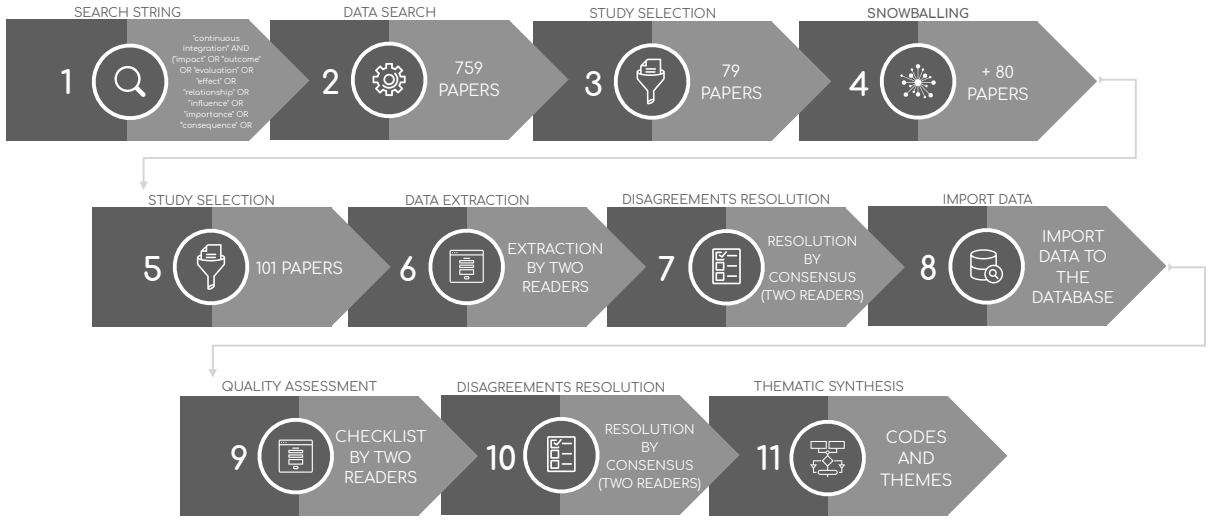


Figure 3 – Research methodology. Step 1: Search string definition; Step 2: Data search; Step 3: Study selection; Step 4: snowballing; Step 5: snowballing study selection; Step 6: Data extraction; Step 7: Disagreements resolution; Step 8: Database import; Step 9: Quality Assessment; Step 10: Quality assessment disagreements resolution; Step 11: Thematic synthesis.

3.1.2 Search strategy

The search process of our SLR consists of the first six steps shown in Figure 3. Step 1–Definition of the search string (section 3.1.2); Step 2–Delimitation of the search mechanisms (section 3.1.2); Steps 3 to 5–Papers screening (section 3.1.3.2).

Search String. Our goal is to find studies that evaluate continuous integration and find the pros or cons of adopting CI in any software development activity. Therefore, we use generic words that express the act of evaluating CI. We craft a string to fetch papers containing the term "continuous integration" and another word that expresses "impact" or "effect" in the title, abstract, or keywords. The terms we used were:

1. "continuous integration"
2. ("impact" **OR** "outcome" **OR** "evaluation" **OR** "effect" **OR** "relationship" **OR** "influence" **OR** "importance" **OR** "consequence" **OR** "study")

Items 1 and 2 were combined with a boolean operator "AND" to match studies with both item 1 and at least one term from item 2. In this way, our search is denoted by the logical expression:

- 1 AND 2

To operationalize our search string in different search engines, we first perform our search using only item 1. Once the results are obtained, i.e., papers containing "continuous integration" on the title, abstract, or keywords, we use scripts to filter out papers not satisfying item 2. The scripts used in this process are available in our digital appendix (SOARES et al., 2021).

Data Search. Regarding the selection of digital libraries, we considered Chen et al. (LERO, 2010) recommendations and included the main publishers' sites and one index engine. Table 2 shows the number of papers we retrieved from each digital library. We apply the search string in each digital library separately and store the results in spreadsheets. As a result, our first search (i.e., step 2 from Fig. 3) resulted in 759 papers.

Table 2 – The digital libraries included in our search along with the number of matches (before and after removing duplicates).

Database	**# of matches**	**%**	**# without duplicates**	**%**
IEEE Xplore	169	22.27	130	27.14
ACM Digital Library	121	15.94	117	24.43
SpringerLink	53	6.98	53	11.06
Wiley Online Library	4	0.53	4	0.84
ScienceDirect	12	1.58	12	2.51
SCOPUS	400	52.70	163	34.03
	759		479	

3.1.3 Study Selection

After performing the first search, we proceed with the study selection step. In this section, we present our selection criteria (Section 3.1.3.1) and the process of paper screening (Section 3.1.3.2).

3.1.3.1 Selection Criteria

In this step, we apply the inclusion and exclusion criteria based on our RQs. This step is necessary to aim for relevant papers retrieved from the studied digital libraries. We apply our inclusion and exclusion criteria in steps 3, 4, and 5 (see Figure 3).

As our work aims to collect evidence reported in the literature regarding the effects of continuous integration (CI) on software development, we are interested in finding empirical studies reporting an evaluation of CI projects or CI project settings (e.g., employees and organization characteristics). To maintain rigor in our analyses, our selected studies must meet a minimum set of quality criteria to provide our review with reliable evidence (we present our quality criteria with more details in Section 3.1.5). Given that most international and high-quality research venues in software engineering use English

as their official language (e.g., ICSE, FSE and TSE), we excluded papers not written in English. Our aim for high-quality and international venues in software engineering is also a mechanism to maintain the rigor of our analyses.

Our inclusion criteria are the following: (i) the studies must be empirical primary studies; (ii) be peer-reviewed papers; and (iii) show that CI adoption may (or may not) have an effect on any aspect of software development. Our exclusion criteria are the following: (i) studies must not be duplicates; (ii) studies must investigate the effects of CI instead of proposing a new tool or a new practice for CI; (iii) papers must be written in English. Figure 4 shows the number of papers removed after applying each criterion.

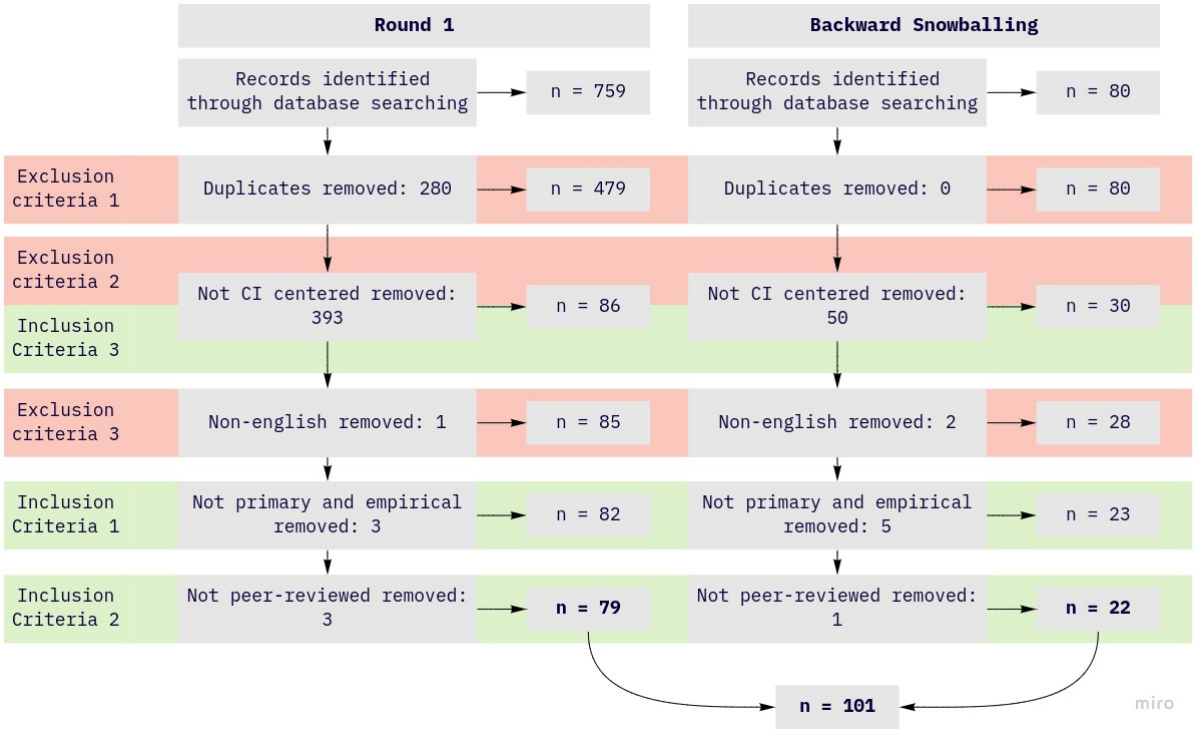


Figure 4 – Diagram illustrating the inclusion and exclusion criteria employment, presenting the number of remaining papers after each stage.

3.1.3.2 Screening of papers

Figure 3 shows an overview of our screening steps. In Steps 1 and 2, we apply our search string (Section 3.1.2) onto the referred digital libraries, obtaining 759 papers. By applying the exclusion criteria 1 in Step 3, we obtain 479 distinct papers (see Table 2 and Figure 4).

In Step 3, we perform a reading of the 479 papers. Two authors read the title and abstract of each study and judged them based on the inclusion and exclusion criteria. By using the Cohen Kappa statistic (COHEN, 1968), we obtain a score of 0.72, which

represents a substantial agreement. Afterward, a third author checks the disagreements (there are 34 disagreements) and resolves each one. As a result, a total of 79 papers were obtained at the end of Step 3.

Since the screening of papers is a step based on explicit inclusion and exclusion criteria, we decided to apply an arbitration disagreement resolution strategy involving a third researcher to check and break a tie in each disagreement. At this step, most of the disagreements (there are 34) are about whether CI is the primary investigation topic of the study. For example, we have an occurrence regarding the study entitled "Moving from Closed to Open Source: Observations from Six Transitioned Projects to GitHub" (P69) on the inclusion criteria 3, since apparently, it does not investigate CI directly. However, the arbiter voted for inclusion, and the paper does present findings on CI.

In the next step (Step 4 in Fig. 3), we perform a backward snowballing, collecting 80 references of the selected studies, which contain the term "continuous integration"—both in the title or abstract. Next, in step 5, two authors read the title and abstract and apply the inclusion and exclusion criteria. At this stage, we include 22 additional papers. We achieved an agreement rate of 0.76 (Cohen Kappa), signaling a substantial agreement between authors. Afterward, we repeated the dispute resolution process with the arbitration of the third researcher, which resulted in 101 studies at the end of Step 5. Figure 4 presents this process in detail with a column for the first cycle — column "round 1", and another for the snowballing process.

Appendix A.2 lists the selected papers. The files containing the lists of papers on each step are available in our digital repository. A backup of the relational database that we use in our SLR is also available (SOARES et al., 2021).

3.1.4 Data Extraction

The extraction process consists of three steps: meta-data retrieval, data extraction, and disagreement resolution. An automated process retrieves the meta-data, which includes the title, authors, year, and publication venue of the studies. We use a reference management tool named Mendeley[1] to support the meta-data extraction. Mendeley exports the meta-data in an XML format. We then use a script to read Mendeley's XML files and store the meta-data into our database.

Two authors extracted data by reading all 101 studies while collecting relevant data (Step 6 in Figure 3). When a paper is completely read by each author, they both submit a form containing the data extracted from that paper. For this purpose, we use a web form containing the fields that are shown in Table 3 (KEELE et al., 2007). Next, we export the data from the forms into a .csv file. Then, we run a script to import the extracted data into our database.

[1] Available at https://www.mendeley.com/

Table 3 – Fields of the extraction form.

	Extraction Form Fields	Sub items
F1	**What are the claims presented in the paper?**	
F2	**What are the variables related to each claim? (And, if it is not clear, what is the meaning of each variable?)**	
F3	**What kind of study was performed to evaluate the claim?**	Controlled Experiments Case Studies Survey Research Action Research MSR
F4	**How the claim was evaluated?**	
F5	**How many projects were involved in the study?**	
F6	**Are there open-source, industry, or both classes of projects involved in the study?**	
F7	**Is the study focused on a specific domain area? which one?**	
F8	**Does the study have the artefacts available?**	
F9.1	**What kind of criteria was considered to determine CI adoption?**	Integration Frequency
F9.2		Automatic Build
F9.3		Build Duration
F9.4		Automated Tests
F9.5		Test Coverage
F9.6		Integration on Master
F9.7		CI SERVICE

Our script automatically checks for the consistency of data provided by the authors. If our script identifies that the two authors extracted different data for a given paper, the script generates a `diff` containing the different content beside each other. The diff files support the resolution of disagreements (Step 7 on Figure 3), in which both authors would check the diff files and reach a consensus regarding which data should be extracted and imported into the database (step 8 on Figure 3).

Examples of inconsistencies include typing errors, misunderstandings of extracting the data, or regarding the study interpretation. Given this interpretative nature, we adopt a consensus disagreement resolution strategy involving both researchers in this step— data extraction. They assess the paper in a virtual meeting to discuss item by item the paper details and then confirm the extracted data in a new form to import.

3.1.5 Quality Assessment

Following the recommendations from Kitchenham & Charters (KEELE et al., 2007), we developed a quality checklist to assess the quality of each of the individual selected primary studies. Our quality assessment aims to understand the quality differences in the collected evidence, supporting the weighting of their claims. Thus, considering the heterogeneity of the selected studies, in terms of study types and the outcomes investigated, we adopt the framework proposed by Dybå et al. (DYBå; DINGSøYR; HANSSEN, 2007). This framework was proposed for the quality assessment of both qualitative and quantitative empirical research.

Therefore, we adapt the Dybå et al. (DYBå; DINGSøYR; HANSSEN, 2007) checklist (see Table 4) composed of 11 questions among four quality criteria: (i) quality of reporting (3 questions — Q2 to Q4); (ii) rigor (4 questions — Q5 to Q8); (iii) credibility (3 questions — Q1, Q9, and Q10); (iv) relevance (1 question — Q11). Questions Q2, Q3, Q5, Q7, Q8, and Q11 are verbatim from Dybå et al. (DYBå; DINGSøYR; HANSSEN, 2007). The remaining five questions were inspired by examples from Kitchenham & Charters (KEELE et al., 2007) and Dybå et al. (DYBå; DINGSøYR; HANSSEN, 2007), maintaining the adequacy to the framework structure.

In this checklist, quality of reporting means the clarity with which it communicates its context, motivation, and goals. The transparency and unambiguity of a study enable readers to extract information and accurate conclusions from it. In this sense, we apply three questions (Q2 to Q4 on Table 4) assessing these issues.

We designated four questions for the rigor criterion (Q5 to Q8 on Table 4), constituting the heaviest factor of this checklist. The questions about rigor highlight the methodological decisions of the studies and their rationale. We analyze whether participants/project selection is suitable or not (e.g., Has the study justified the selection procedures?). We also look for the metrics and measures and if they are provided/explained. We observe if the research design is appropriate to the research goals (e.g., Has the researcher justified the research design? Has the researcher presented and explained the statistical tests applied?). Furthermore, in Q8, we look for comparison or control groups to indicate analytical rigor.

The credibility factor comprises three items (see Q1, Q9, and Q10 on Table 4) assessing acceptability and the coherence between the presented findings and applied methods. The first question filters peer-reviewed approved studies. We ask if empirical data and experiment results support the findings and conclusions (e.g., Are the findings explicit? Are limitations of the study discussed explicitly? Are the findings discussed concerning the original research questions?). Lastly, we check whether data is available (or scripts or detailed descriptions to obtain it) for reproduction or replication.

Finally, we assess the relevance of contributions (see Q11 on Table 4) for industry or academy as an indicator of the study quality. This criterion has the lightest weight. In

question 11, we check if the researchers discuss the impact of their study on the state-of-the-art and state-of-the-practice (e.g., do they consider the findings concerning current practice or relevant research-based literature?).

These 11 questions behave as binary variables (1 - yes; 0 - no), and together, they provide a metric of quality and reliability of the findings. To cover a broad set of empirical evidence and draw a big picture of continuous integration reported effects, just the Q1 was used as an inclusion criterion (section 3.1.3.1). The remaining questions compound a checklist to assess the strength of the body of evidence in Sections 3.2.2 and 3.2.3. Therefore, we only evaluated studies with collected findings in our approach (i.e., papers from which we find claims).

The sum of 11 questions allows us to compute a quality score per study (see Section 3.2.3.3). We consider this score a measure of the reliability of the extracted claims, i.e., claims originated from studies with high scores are more reliable than those with lower scores. We built our checklist with the goal of rewarding a greater variety of methods to support a claim. For example, the value of method variability is clearly seen in the higher scores obtained by mixed-methods studies (MSR and survey). Mixed-methods studies score better than other types with a median of 10 points (see section 3.2.3.3). The overall median score is 9.

Furthermore, we consider certain codes (see section 3.1.6) more reliable if they are supported by a higher number of studies and a higher variety of study types. In sections 3.2.2 and 4.3, we assess and discuss CI claims by examining: the set of studies supporting these claims, the variety of methods to support these claims, the complementarity between findings, and the respective quality scores of the studies. For example, in section 3.2.2.4, we present a code describing an association between CI and a *"change in commit patterns."* This code represents five claims over three studies with various methods and quality scores (one Case Study, one Mining Software Repository — MSR, and one MSR/Survey). Although the case study scores 6 points of quality, the MSR scores 9 points, and the MSR/Survey scores 10 points, i.e., different methodologies combined with an overall higher quality score support the claim that CI promotes a *"change in commit patterns."* Therefore, this code is more reliable than if it was supported by only a case study or other studies of the same type and similar quality scores.

Similar to what was exposed in section 3.1.4 for data extraction, after reading, two authors independently assessed the quality of the study using a web form, achieving a Kappa score of 0.55, which indicates a moderate agreement. To subsidize this quality assessment, we added guiding questions for each item in the quality checklist. Later, each divergence was discussed between the pair and settled by consensus after revisiting the study (step 10 in Figure 3).

Table 4 – Quality Assessment checklist

	Question
Q1	Was the paper peer-reviewed?
Q2	Is there a clear statement of the aims of the research?
Q3	Is there an adequate description of the context in which the research was carried out?
Q4	Is the size of the data set stated?
Q5	Was the recruitment strategy appropriate to the aims of the research?
Q6	Are the definitions for the measures or metrics provided?
Q7	Was the research design appropriate to address the aims of the research?
Q8	Is there a comparison or control group?
Q9	Does the empirical data and results support the findings?
Q10	Is the data available?
Q11	Is the study of value for research or practice?

3.1.6 Synthesis

In step 11 of Figure 3, we use the data extracted from our extraction form (Table 3) to address RQ1, RQ2, and RQ3 (Section 3.1.1). We first analyze the demographic data (see Appendix A.1). Next, we perform the analyses to answer the Research Questions.

To answer *RQ1–What are the existing criteria to identify whether a software project uses CI?*, we use the F9 field. To answer *RQ2–What are the reported claims regarding the effects of CI on software development?*, we run a *thematic synthesis* (CRUZES; DYBå, 2011) using the fields F1 and F2. To answer *RQ3–Which empirical methods, projects and artifacts are used in the studies that investigate the effects of CI on software development?*, we use fields from F3-to-F8 (see Table 3).

In the *thematic synthesis* to answer RQ2, we follow the steps recommended by Cruzes & Dybå (CRUZES; DYBå, 2011). The thematic synthesis consists of identifying patterns (themes) within the data, which provides a systematic manner to report the findings of a study. The thematic synthesis consists of five steps:

1. Extract data,
2. Code data,
3. Translate codes into themes,
4. Create a model of higher-order themes, and
5. Assess the trustworthiness of the synthesis.

The *Extract Data* is the first step of the thematic synthesis (Section 3.1.4). To answer RQ2, we analyze the data from fields F1 and F2 (see Table 3), which are *claims regarding the effects of CI*, i.e., any consideration in a study indicating a positive or negative effect of CI on the software development phenomena. Therefore, we do not consider to

be a *claim* statements that are indirect or unrelated to the effects of CI on software development—even if CI is used by the software projects under investigation. Table 5 shows two examples of claims.

Table 5 – Extracted claims from studies P25 and P74, from fields F1 and F2 of the extraction form.

Claim	**Variables**	**Paper id**
CI increases normalized collaboration amount between programmers (OSS and proprietary projects)	Normalized median in-degree (NMID)	25
Core developers in teams using CI are able to discover significantly more bugs than in teams not using CI.	Number of bug reports (i.e. issues clearly labeled as bugs)	74

We group the information from fields F1 and F2 in a spreadsheet. Next, we code the information through an inductive approach (CRUZES; DYBå, 2011), i.e., two authors analyze all the *claims* together and collaboratively assign one or two codes to each of the claims. The assigned codes are based on the central message within a claim. Therefore, the two authors create an established *list of representative codes*.

Once the list of codes is created, two other authors are debriefed regarding the codes to understand their meanings. These two other authors revisit every claim independently and select one or more codes from the list of codes to assign to the claims. As an example, consider the following finding in study P25: "After the adoption of CI, normalized collaboration amount between programmers significantly increases for our set of OSS and proprietary projects. [...]". Both authors assign the code *"CI IS ASSOCIATED WITH AN INCREASE IN COOPERATION"* to such a claim. At this stage, we obtain a Cohen Kappa statistic (COHEN, 1968) of 0.73, which indicates a substantial agreement.

At this step, since it is a task of synthesizing ideas, we adopt an arbitration disagreement resolution strategy to explore contributions from a more experienced author. All disagreements were solved by a third author. As an example of disagreement, consider the following claim in study P74: "Core developers in teams using CI are able to discover significantly more bugs than in teams not using CI. [...]". One author assigned the code *"CI IS ASSOCIATED WITH DEFECT REDUCTION"*, while the other author assigned *"CI IS ASSOCIATED WITH A DECREASE IN TIME TO ADDRESS DEFECTS"*. In this case, the third author analyzed the claim and decided to maintain the code *"CI IS ASSOCIATED WITH DEFECT REDUCTION"*.

In the third step of the thematic synthesis (i.e., *Translate codes into themes*) we compute the frequency of each code and propose overarching themes. Finally, we develop a thematic network to express the relationship between codes and themes (Step 4 of the thematic synthesis). Once the thematic network was developed we performed two meetings with all authors to discuss the meaningfulness of the network and codes (Step 5 of the

thematic synthesis). After 4 hours of discussion (each meeting having 2 hours), we refined the thematic network and the codes and themes within it (see Section 3.2.2 and Figure 6).

3.2 Results

The appendix A.1 present some demographic information about the studies. In this Section we present the results of our systematic literature review (SLR). The following subsections explores the results to our research questions.

3.2.1 RQ1: What are the existing criteria to identify whether a software project uses CI?

To answer this research question, we analyze in the primary studies which criteria (e.g., CI practices or attributes) were considered when describing or selecting the analyzed projects—For example, how are the projects using CI deemed as such? More specifically, we do not investigate the analyzed projects themselves. Instead, we investigate whether our primary studies select (or classify) their analyzed projects based on the following criteria: integration frequency, automated build, build duration, automated tests, test coverage threshold, integration on the mainline, and CI service (DUVALL, 2013; FOWLER, 2020).

As discussed in section 2 there is no consensus around the definition of CI or a homogeneous set of CI practices (DUVALL, 2013; FOWLER, 2020; STåHL; BOSCH, 2014b; ZHAO et al., 2017; SHAHIN; BABAR; ZHU, 2017; FITZGERALD; STOL, 2014; VIGGIATO et al., 2019). Therefore, considering the literature diversity, we adopt a set of criteria based on Duvall's seven cornerstones (DUVALL, 2013), and CI practices highlighted by Fowler (FOWLER; FOEMMEL, 2006) for this analysis as they present a prescriptive minimal number of practices, instead of discussing the variability among diverse CI implementations.

Figure 5 (a) shows the number of criteria considered in the primary studies to identify whether a project uses CI. From the seven considered criteria that we expect to see, 43 (42.5%) of the primary studies, surprisingly, did not apply or determine any of them. On the other hand, 26 (25.7%) of the primary studies used two criteria, while 16 (15.8%) of the studies and another 16 (15.8%) of them used one and three criteria, respectively.

By inspecting the 43 (42.5%) studies without clear criteria for determining whether a project uses CI, we observe that: (i) 28 of the studies do not analyze data related directly to the project's development. Instead, they are studies based on interviews, surveys, companies, or other data sources, e.g., build logs; (ii) some of the studies present experience reports without further details regarding how projects adopt the CI practices. In addition, (iii) a few studies (P49, P52, P60, P69) analyze both projects and self-described

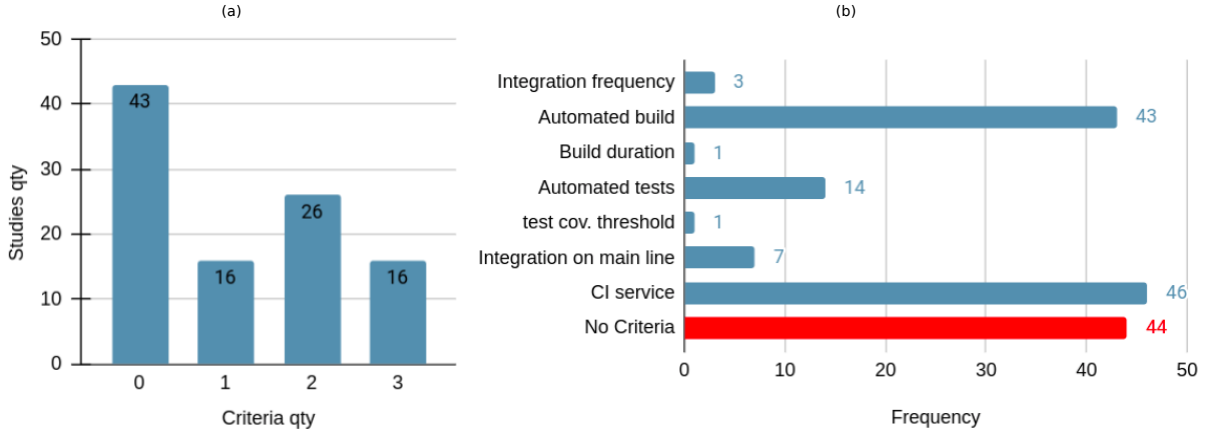


Figure 5 – (a) Histogram representing the proportion of primary studies using a number of CI criteria; (b) Frequency of usage of each criteria;

declarations, like interviews or surveys.

Although it is understandable that the criteria we are looking for (e.g., integration frequency) may not be applied in such studies, it would still be valuable to perform certain checks during the interviews or surveys. For example, questions such as *"on a scale of 1 to 7, how would you classify that your project adheres to CI?"* along with a definition of CI could help such studies to gauge the quality the CI practices that are implemented by the subjects. Regarding the studies that investigate build logs only (e.g., build logs from TravisCI), it would also be desirable to be more restrictive regarding these logs, since not every build log from TravisCI may come from a project that properly employs CI. Therefore, solely relying on the fact that build logs are generated from a CI server does not necessarily imply that the derived observations can be associated with the adoption of CI practices.

Concerning studies applying only one criterion to identify whether a project uses CI, the CI server configuration is the most common criterion (9/16 studies - 56,25%). We observe in Figure 5 (b) that the usage of a CI service is the most common criterion applied. This criterion consists of checking whether subject projects have used a CI service (e.g., TravisCI). The second most frequent criterion is checking whether subject projects perform automatic builds. Table 6 shows the CI services cited in the included studies, revealing that TravisCI [2] is the most used CI service, confirming the finding by Hilton et al. (HILTON et al., 2016).

[2] https://travis-ci.com/

Table 6 – CI Services cited in the included studies.

CI Services	Studies
TRAVIS CI	37
JENKINS	8
CUSTOMIZED	4
CIRCLE CI	3
APPVEYOR, TEAM CITY, WERKCER	2
BUILDBOT, CONCOURSE, CRUISECONTROL, GERRIT, CLOUDBEES, XCODE BOTS, GITLAB	1

3.2.2 RQ2: What are the reported claims regarding the effects of CI on software development?

To answer RQ2, we collect the claims from our primary studies and proceed with the thematic synthesis to produce the codes and themes regarding the claims. As explained in Section 3.1.6, a claim is a statement regarding any positive or negative effect of CI on the software development phenomena. We found 125 claims regarding the effects of CI in 38 out of 101 papers (37.6% of studies). From the thematic synthesis, we produce 31 codes from the 125 extracted claims. Figure 6 shows the produced codes organized into 6 overarching themes.

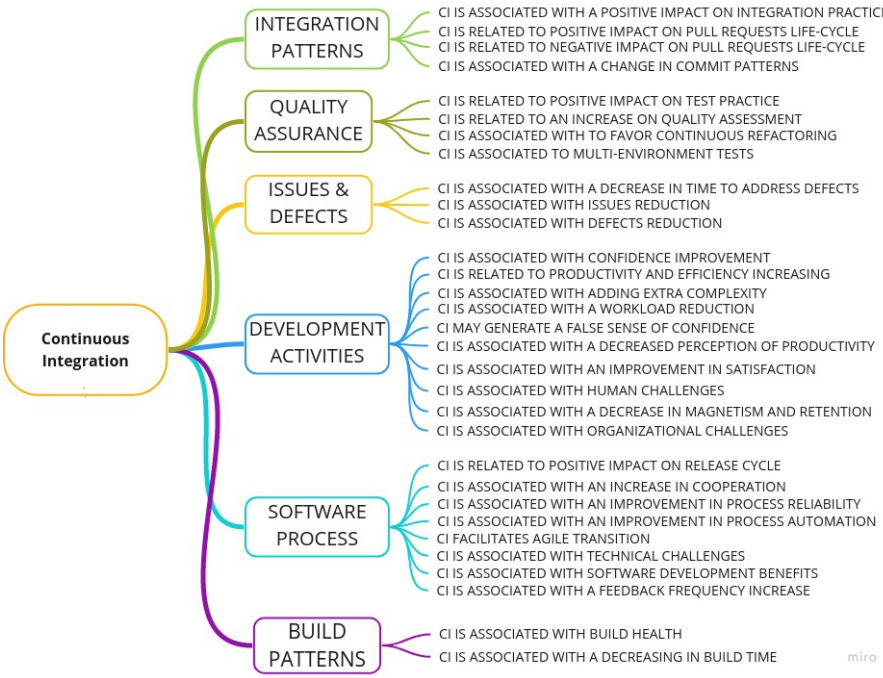


Figure 6 – Themes and codes representing the studies claims.

Table 7 shows the following information: a) the themes, b) the number of claims pertaining to a theme, and (c) the primary studies that make the claims. The most common themes in the primary studies are: *"development activities"*— having 35 claims across

18 papers— and *"software processes"*— with 35 claims across 18 papers. Although we observe in RQ1 that automated builds is a common criterion to check whether CI is used by subject projects, the theme *"build patterns"* has only 7 claims from 4 primary studies.

Table 7 – Number of claims and studies that pertain to a theme.

Theme	Number of Claims	Primary Studies
Development activities	35	P7, P9, P31, P39, P52, P58, P59, P73, P79, P74, P90, P91, P93, P97, P99, P100, P102, P106
Sofware process	35	P4, P24, P25, P38, P40, P46, P52, P58, P64, P79, P81, P92, P93, P97, P100, P102, P105, P106
Quality assurance	23	P14, P29, P44, P52, P58, P64, P79, P89, P93, P97, P100, P102, P106
Integration patterns	22	P25, P47, P59, P69, P74, P79, P81, P89, P97, P100, P102, P104
Issues & defects	14	P25, P31, P50, P59, P74, P79, P97, P100, P89, P106
Build Patterns	7	P29,P49, P97, P102

The link between a theme and a paper does not necessarily mean that the theme is the paper's main topic. A paper may have one or more claims related to a theme, but the same paper may have other claims related to other themes. Figure 7 shows a conceptual class diagram expressing how a study can have none or several claims, while each claim can be related to one or more codes. Each code is related to a theme. Section 3.1.6 describes the entire process of our synthesis. Therefore: (i) each code sentence shown at the right side of Figure 6 is representative of a set of claims extracted from primary studies and mapped to such code; (ii) a study is not necessarily linked directly to a theme but may be related to several themes by transitivity.

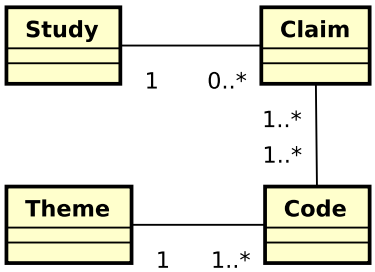


Figure 7 – Conceptual class diagram of relationships between studies, claims, codes, and themes.

To further evaluate the reliability of the findings within the themes, we add an earlier discussion about quality scores ($Qscore$) in the subsequent Section 3.2.3. For the purpose of our analysis, we consider the mean ($\bar{Qscore}$) and the median ($\tilde{Qscore}$) as the quality measurement for our body of evidence.

3.2.2.1 Development Activities

Table 8 – Codes from the "development activities" theme. We show the number of claims related to the code, the primary studies supporting it, the mean and median of quality scores of such studies.

Code	Number of Claims	Primary Studies	$\bar{Qscore}$	$\tilde{Qscore}$
CI is related to productivity and efficiency increase	12	P31, P39, P52, P59, P73, P79, P74, P97, P100, P102, P106	8.8	8
CI is associated with Confidence improvement	6	P39, P58, P93, P97, P100, P106	7.0	7
CI is associated with adding extra complexity	5	P7, P58, P106	6.2	7
CI may generate a false sense of confidence	3	P58, P106	6.0	7
CI is associated with a workload reduction	3	P6, P79, P93	5.6	5
CI is associated with human challenges	2	P58, P59	5.5	4
CI is associated with a decreased perception of productivity	1	P91	8.0	8
CI is associated with an improvement in satisfaction	1	P9	8.0	8
CI is associated with organizational challenges	1	P99	9.0	9
CI is associated with a decrease in magnetism and retention	1	P90	9.0	9
		Overall	**7.1**	**7**

Several primary studies have claims regarding the effects of CI on development activities. Table 8 shows the claims, the development activity related to the claim, the ID of the primary studies, the mean $\bar{Qscore}$ and the median $\tilde{Qscore}$. We observe 4 positive effects of CI on productivity, efficiency, confidence, satisfaction, and reduction in the workload.

There are several mentions in the primary studies claiming an increase in productivity and efficiency when using CI (12 occurrences in 11 studies). As reported in study P97:

> *"According to our interview participants, CI allows developers to focus more on being productive, and to let the CI take care of boring, repetitive steps, which can be handled by automation."* (p. 203)

And

> *"Another reason [...] was that CI allows for faster iterations, which helps developers be more productive."* (p. 204)

Several studies also mention an improvement in confidence after using CI (6 occurrences in 6 papers). Paper P58 states the following:

> *"Depends on the coverage, but some sort of confidence that introduced changes don't break the current behavior. [...] I assume the most critical parts of the system have been covered by test cases."* (p. 76)

These positive associations with CI have the support of numerous and diverse studies reported in Table 8 with a substantial quality profile. However, there are still low scores on the rigor criterion in our quality assessment (Q5 to Q8 on Table 4). For example, regarding CI increasing productivity and efficiency, P79 does not substantiate its assumptions with statistical tests, and P106 does not apply any control or comparison group (Q7 and Q8 of the quality checklist). In addition, 5 out of 11 studies scored 2 or fewer points on the rigor criterion (out of 4 points). Regarding "confidence improvement", P58 and P106 do not satisfy Q7, and 4 out of 6 studies scored 2 or fewer out of 4 points on the rigor criterion.

The association of CI with a workload reduction has low-quality scores of $Q\bar{s}core = 5.6$ and $Q\tilde{s}core = 5$, and the association of CI with an improvement in satisfaction has only one supporting study, although with a high-quality profile ($Qscore = 8.0$).

Nevertheless, not everything seems to be positive in terms of development activities. We found six negative aspects stated in the primary studies: extra complexity added, the existence of a false sense of confidence, human and organizational challenges, a decreased perception of productivity, and a decrease in magnetism and retention of collaborators in projects. The most endorsed negative effects of CI are the addition of extra complexity (5 occurrences in 3 papers), and the generation of a false sense of confidence (3 occurrences in 2 studies). P7 mentions the extra complexity related to using CI:

> *"Results of our study [...] highlights the complexity of dealing with CI in certain situations, e.g., when dealing with emulated environments, non-deterministic (flaky) tests, or different environments exhibiting an inconsistent behavior".* (p. 47)

P106 explains the false sense of confidence:

> *"As opposed to the confidence benefit, respondents described the false sense of confidence as a situation of which developers blindly trust in tests"* (p. 2232)

However, some of these negative effects obtain low-quality scores, especially on the rigor criterion (e.g., P58 scored 0 out of 4 points), and need to be further investigated by our research community. Regarding the increase in human challenges, for example, studies obtain scores of $Q\bar{s}core = 5.5$ and $Q\tilde{s}core = 4$ (see Table 8). The claims suggesting a "false sense of confidence" as an effect of CI come from only two studies conducted by the same authors.

On the other hand, other negative associations, such as "CI is associated with a decreased perception of productivity", "CI is associated with organizational challenges",

and "CI is associated with a decrease in magnetism and retention" obtain the highest quality scores. However, all these negative associations are supported by only one study each. The association "CI is associated with adding extra complexity" obtained scores of $Q\bar{s}core = 6.2$ and $Q\tilde{s}core = 7$, particularly one study — P7 — conducted a mixed-method study (MSR + Survey) and has a high quality score (Qscore = 9).

These adverse effects seem to point in the same direction: CI introduces complexity, challenges the organizational environment, and influences developers' perception of productivity. Such statements contradict the assumption that CI promotes a workload reduction. We argue that more studies are required by our community to better understand the context and the extent of such adverse effects, given the small variety and generalizability of studies supporting them. Furthermore, the studies' overall rigor mean is near 50% of the max score. This result suggests that we need more efforts in performing reliable studies on the *Development Activities* theme.

> According to the literature, there is reliable evidence of the association between CI and improved productivity, efficiency, and developer confidence. CI may create a positive impact on the stakeholders' satisfaction. On the other hand, findings suggest that CI introduces complexity to the development environment, demanding more developer effort and discipline, negatively impacting developers' perception of their productivity. Given the low number of studies related to some of these evaluation aspects, there is room for further studies on these aspects.

3.2.2.2 Software Process

Table 9 – Codes from theme "Sofware Processes". We show the number of claims related to the code, the primary studies supporting it, the mean and median of quality scores of such studies.

Code	Number of Claims	Primary Studies	$Q\bar{s}core$	$Q\tilde{s}core$
CI is related to positive impact on release cycle	7	P58,P64, P97, P100, P102	7.8	10
CI is associated with an increase in cooperation	8	P25, P52, P79, P81, P93, P106	8.0	8
CI is associated with an improvement in process reliability	7	P46, P52, P79, P92, P105, P106	7.8	8
CI is associated with technical challenges	6	P4, P24, P38, P58, P97	6.8	6.5
CI is associated with an improvement in process automation	3	P58, P97, P106	7.0	7
CI is associated with software development benefits	3	P40, P92, P105	9.0	9
CI is associated with an increase in the feedback frequency	1	P46	5.0	5
CI facilitates the transition to agile	1	P46	5.0	5
		Overall	**7.5**	**8**

In this theme, we group codes related to the effects of CI on the software processes. Table 9 presents the codes, the number of claims supporting the code, the primary studies in which they appear, the mean ($Q\bar{s}core$) and median ($Q\tilde{s}core$) scores. We map seven codes representing findings of positive effects of CI, and one negative effect.

Three studies (see the quality scores in Table 9) claim that "CI is associated with software development benefits", being considered a factor of success and contributing to a decrease in the rate of project failure. Moreover, "CI is associated with an improvement in process reliability" in six different studies pointing out progress in transparency, stability, predictability, and support for a quantitative view of progress. These studies are diverse in their claims, and there is room to further investigate what benefits and how CI contributes to process reliability. Moreover, some of these studies did not perform well in our quality assessment. P46 obtains a $Qscore = 5$, P79 a $Qscore = 8$ and P106 a $Qscore = 7$, all of them have issues with the rigor criterion. For example, P106 scored 1 out of 4 on the rigor criterion, and P79 only shows descriptive statistics as a means to support its claims.

There is evidence that "CI is associated with an increase in cooperation", e.g., improving inter-team and intra-team communication (P52). Regarding this association, there are three studies with a low $Qscore$, due to the rigor criterion. For example, P79 does not ground its claims on statistical tests. Also, P93 and P106 obtain 0 out of 4 points on the rigor criterion. On the other hand, P25, P52, and P81 convey methodological confidence. These studies perform well in all quality criteria and provide more reliability to the association between CI and increased cooperation. Concerning cooperation, the primary study P25 states:

> *"After adoption of CI, normalized collaboration amount between programmers significantly increases for our set of OSS and proprietary projects."*. (p. 12)

Some studies still suggest "an improvement in process automation", regarding this, the P97 discusses:

> *"CI allows developers to focus more on being productive, and to let the CI take care of boring, repetitive steps, which can be handled by automation."*. (p. 203)

The increase in automation mentioned by P97 is believed to lower the developers' workload. However, the increase in automation also introduces technical challenges (shown below) to the development process that are associated with a perceived decrease in productivity (see Section 3.2.2.1).

Five studies shed light on the relationship between CI with and a "positive impact on release cycle". Such studies show that continuous integration promotes fast iterations supporting fast and regular releases. For example, P100 states:

> *"We found that projects that use CI do indeed release more often than either (1) the same projects before they used CI or (2) the projects that do not use CI.".* (p. 432)

We can note in Table 9 a significant variety and high-quality scores (*Qscore*) of the studies claiming an association between CI and positive impacts on the release cycle, cooperation, process reliability, and software development benefits. On the other hand, few studies suggest that CI encourages process automation, and the codes "CI is associated with an increase in feedback frequency" and "CI facilitates the transition to agile" are presented by just one study, which is P46, with $Qscore = 5$ (below the average).

A potential negative effect of CI in the *Software Process* are the technical challenges associated with adopting CI (6 occurrences in 5 studies), confirming the addition of extra complexity (see 3.2.2.1). For example, study P58 states:

> *"As regarding the hidden problems associated with continuous integration usage, we found that 31 respondents are having a hard time configuring the build environment".* (p. 76)

Conversely, it is essential to highlight the low scores obtained by some studies supporting the association with technical challenges. For example, P38 and P58 obtained low scores mainly because of the rigor criterion (scores of 0 out of 4). This result suggests poor methodological reliability from these studies. On the other hand, P4 and P97 obtained a high *Qscore* in all aspects and support the association between CI and technical challenges. Overall, studies highlight the difficulty in implementing CI as well as setting up the environment, especially to newcomers (by confirming problems of magnetism and retention of developers, see Section 3.2.2.1). Moreover, some studies state that the lack of maturity of technology may contribute to the abandonment of CI, as stated in study P24:

> *"Results show that all of the 13 interviewees mentioned challenges related to tools and infrastructure such as code review, regression feedback time when adopting to CI. The maturity of the tools and infrastructure was found to be a major issue."*(p. 29)

These studies are diverse, being two case studies, two surveys, and one MSR/Survey, with quality scores of $\bar{Qscore} = 6.8$ and $\tilde{Qscore} = 6.5$. As in the previous theme, the studies of this theme also highlight the human challenges and the extra complexity added, i.e., CI adds some level of complexity to practitioners (see Section 3.2.2.1). Additionally, the technical challenges seem to be related to CI configurations and seem to impact newcomers and specific domains (such as embedded systems). P58, for example, argues that newcomers may face barriers to create a build due to a lack of experience with the

project. In turn, P38 shows that some embedded systems contain complex user scenarios that require manual testing.

> CI is mentioned as a success factor in software projects, positively affecting software processes, promoting faster iterations, more stability, predictability, and transparency. However, CI may also bring technical challenges to the team related to the build environment and tools. Practitioners and researchers may consider such challenges and elaborate strategies to mitigate them. Moreover, there is still space for studies to answer questions about automation and productivity.

3.2.2.3 Quality Assurance

Table 10 – Codes from the "Quality Assurance" theme. We show the number of claims related to the code, the primary studies supporting it, the mean and median of quality scores of such studies.

Code	Number of Claims	Primary Studies	$\bar{Qscore}$	$\tilde{Qscore}$
CI is related to positive impact on test practice	10	P14, P29, P52, P89, P93, P97, P100, P102, P106	8.6	9
CI is related to an increase on quality assessment	8	P44, P58, P64, P79, P97, P106	6.7	7
CI is associated with to favor continuous refactoring	1	P44	9.0	9
CI is associated to multi-environment tests	4	P58, P97, P100, P106	7.7	7
		Overall	**7.8**	**9**

Another significant theme that emerged from our primary studies is "Quality Assurance". As shown in Table 10, in general, CI is associated with continuous practice of quality assessment (P44), refactoring (P44), finding problems earlier (P58), and improving the code quality (P58, P64, P97, P106). These associations emerge from the perception that CI can be used as a quality assessment, providing transparency and supporting multi-environment tests. One of these studies (P44) also suggests that CI provides an adequate context for employing continuous refactoring.

Under the code "CI is related to an increase on quality assessment", there is little diversity of study types. In addition, P58 ($Qscore = 4$) and P64 ($Qscore = 5$) have low $Qscores$. On the other hand, there is a convergence between the studies regarding greater awareness of code and product quality (P44, P58, P64, P79, P97, P106).

Other studies with a wider variety of methods and high-quality scores provide reliable support for the association between CI and good test practices (at least in terms of the number of tests and test coverage). The exceptions are studies P58, P93, and P106 with a low $Qscore$, especially regarding the fragility in validating their claims in their

respective data. However, although P58, P93, and P106 obtain low scores, the median *Qscore* for the association between CI and good test practices is still strong.

According to the primary studies, the adoption of CI tends to enforce automated software testing, increasing the volume and coverage of tests. CI also encourages best practices of automated tests ranging from tests within private builds to functional tests on the cloud. The support to multi-environment tests is also mentioned as a support to a "real-world environment". For example, study P89 states:

> *"After some (expected) initial adjustments, the amount (and potentially the quality) of automated tests seems to increase."* (p. 69)

In the same way, study P97 states:

> *"Developers believe that using CI leads to higher code quality. By writing a good automated test suite, and running it after every change, developers can quickly identify when they make a change that does not behave as anticipated, or breaks some other part of the code."* (p. 203)

CI is perceived as a provider of transparency and continuous quality assessment through enforcing the test practices and supporting multi-environmental tests. There is reliable evidence on the association between CI and test increasing and coverage. There is room to further investigation on the test quality and test effort in CI projects.

3.2.2.4 Integration Patterns

Table 11 – Codes from the "Integration Patterns" theme. We show the number of claims related to the code, the primary studies supporting it, the mean and median of quality scores of such studies.

Code	Number of Claims	Primary Studies	$\bar{Qscore}$	$\tilde{Qscore}$
CI is related to positive impact on pull request life-cycle	10	P47, P69, P74, P81, P89, P100, P104	10.0	10
CI is associated with a commit pattern change	5	P25, P89, P102	8.8	9
CI is associated with a positive impact on integration practice	5	P59, P79, P97, P100, P102	8.2	8
CI is related to negative impact on pull request life-cycle	3	P81, P89	10.0	10
		Overall	**9.3**	**10**

The "Integration Patterns" theme consists of claims related to commits and pull requests, as shown in Table 11. The association between CI and a "positive impact on integration practice" is observed by five studies through three case studies (P59, P79, P102)

and two surveys (P97, P100). There are mentions of CI as a facilitator to the integration practice, making the code integration easier (P97, P100), reducing the stress (P59), and supporting a faster integration (P79, P102). Such benefits could also be a motivation for CI adoption, as the authors of P100 explains:

> *"One reason developers gave for using CI is that it makes integration easier. One respondent added 'To be more confident when merging PRs.' "* (p. 433)

The CI association with "a change in commit patterns", i.e., the way developers commit, can confirm the perceived benefits through three studies: one MSR (P25 - $Qscore = 9$), one MSR/survey (P89 - $Qscore = 10$), and one case study (P102 - $Qscore = 6$). When analyzing the evolution of projects, studies identify an increasing frequency and a decreasing size of commits. For example, study P89 applied an *Regression Discontinuity Design (RDD)* associated with a survey to understand the longitudinal effect of CI adoption (TRAVISCI adoption) and found: an increasing number of merge commits as an indicator of a workflow change (e.g., migration to a pull-based model); and a decrease in size of merge commits as an indicator of more frequent integration.

Some primary studies with high *Qscore* and high variety of methods report a "positive impact on pull request life-cycle", such as (i) an increase in the number of pull requests (PRs) submissions (P81); (ii) an increase in the number of PRs closed (P89); (iii) acceleration in the integration process (P100); (iv) higher support to identify and reject problematic code submissions more quickly (P47, P69, P74, P100); (v) a higher contribution from external collaboration (P74); (vi) a higher delivery of PRs (P81). However, these studies also reveal a "negative impact on pull request life-cycle", i.e., merging pull requests might become slower after adopting CI (P81, P89). P81 states that:

> *"Open source projects that plan to adopt CI should be aware that the adoption of CI will not necessarily deliver merged PRs more quickly. On the other hand, as the pull-based development can attract the interest of external contributors, and hence, increase the project's workload, CI may help in other aspects, e.g., delivering more functionalities to end-users."* (p. 140)

The study P81 also indicates that CI can be associated with an increase in the lifetime of pull request:

> *"We observe that in 54% (47/87) of our projects, PRs have a larger lifetime after adopting CI.".* (p. 134)

We discuss this apparent contradiction in more details in Section 3.3.2.1.

Although the three case studies P59 ($Qscore = 7$), P79 ($Qscore = 8$), and P102 ($Qscore = 6$) obtain lower *Qscores* than the rest of studies in the *Integration Patterns* theme, the variety of methods and the overall mean quality of the studies in this theme is

high — $\bar{Qscore} = 9.3$. Additionally, the studies seem reliable as their overall rigor mean is of 2.83 (out of 4 points).

> The studies report a perception of CI as a facilitator to the integration practice, influencing positively the way developers perform commit even the workflow. CI can benefit the pull-based development by improving and accelerating the integration process. However, there is evidence that CI may increase the lifetime of pull requests.

3.2.2.5 Issues & defects

Table 12 – Codes from the "Issues & Defects" themes. We show the number of claims related to the code, the primary studies supporting it, the mean and median of quality scores of such studies.

Code	Number of Claims	Primary Studies	$\bar{Qscore}$	$\tilde{Qscore}$
CI is associated with defect reduction	6	P25, P31, P50, P74, P79	8.5	9
CI is associated with issues reduction	2	P25, P89	9.5	9
CI is associated with a decrease in time to address defects	6	P50, P59, P79, P97, P100, P106	8.6	8
		Overall	**8.7**	**9**

The manner by which CI projects address defects, bugs, and issues is grouped under the "issues and defects" theme. Table 12 shows three codes representing these aspects and summarizes their occurrences. These studies consistently indicate that CI enables teams to detect and address issues earlier, which is related to an overall decrease in the number of issues and defects reported, i.e., an external quality improvement. For instance, study P50 states that:

> *"The descriptive statistics point to an overall improvement in not only finding more defects (defect reduction), but also in shortening the time required to fix the defects (defect lead and throughput)."* (p. 8)

We observe other statements regarding CI helping development teams to find and fix bugs and broken builds, shortening the time to fix these bugs and builds. In particular P59 states:

> *"An indirect, but important, advantage of CI is related to the following human factor: the earlier the developer is notified of an issue with the patch that was just committed, the easier it is for him or her to associate this regression with specific changes in code that could have caused the problem and fix it."* (p. 9)

However, regarding issues or bugs resolution rate, while study P25 identified an increase in the resolution rate after CI adoption for OSS projects, P89 identifies that issue resolution tends to be slower after CI adoption.

The overall mean quality score of studies in this theme is high — $\bar{Qscore} = 8.7$ and $\tilde{Qscore} = 9$, having also an overall mean rigor of 2.6 (out of 4 points). The quality weaknesses are compensated by other studies with higher quality scores and methodological rigor. For example, for the code "CI is associated with a decrease in time to address defects", P79 and P106 claim that CI supports the team to catch issues earlier. Similarly, P50, P59, P97, and P100 corroborate the claim that CI helps to find and fix problems earlier. Regarding the code "CI is associated with defect reduction", P31 and P79 reveal an association with reduced defects, which P50 confirms. Additionally, P74 claims that CI yields a higher bug discovery, and P25 claims that CI yields a higher bug resolved rate.

Studies suggest that CI can improve the time to find and address issues. They also observed a decrease in defects reported.

3.2.2.6 Build Patterns

Table 13 – Codes from the "Build Patterns" theme. We show the number of claims related to the code, the primary studies supporting it, the mean and median of quality scores of such studies.

Code	Number of Claims	Primary Studies	$\bar{Qscore}$	$\tilde{Qscore}$
CI is associated with build health	6	P29,P49, P97, P102	8.0	7
CI is associated with a decreasing in build time	1	P102	6.0	6
		Overall	**7.7**	**7**

The "Build Patterns" theme encompasses the reported associations between CI and build metrics. Table 13 shows two codes representing the "Build Patterns" theme. Under the code "CI is associated with build health", 6 mentions in 4 studies report developers' good practices encouraged by CI, as well as an improvement in build success rate (P49, P102). CI encourages good practices that contribute to build health, such as testing in private builds (P29), prioritization to fix broken builds (P29), and supporting a shared build environment (P97). P97 states the following:

> *"Several developers told us that in their team if the code does not build on the CI server, then the build is considered broken, regardless of how it behaves on an individual developer's machine. For example, S5 said:'...If it doesn't work here (on the CI), it doesn't matter if it works on your machine."'* (p. 202)

One study, P102, also reports a decrease in the build time. On the other hand, P97 registers long build time as a common barrier faced by CI developers. Therefore, we argue that there is scope for further investigation of the factors that influence build time, build health, and relationships with other variables in CI projects. For example, only one of these studies analyzes build data (P49) in a controlled context. In addition, the overall mean quality score of studies in this theme is not high — $\bar{Qscore} = 7.7$ and $\tilde{Qscore} = 7$, with an overall mean rigor of 2.5 (out of 4 points).

> CI promotes good practices related to build health and contributes to an increase in successful builds.

3.2.3 RQ3: Which empirical methods, projects and artifacts are used in the studies that investigate the effects of CI on software development?

In this RQ we investigate the methodologies applied in the primary studies. In particular, we analyze: the kind of projects that our primary studies analyze (Section 3.2.3.1), the study methodologies, i.e., the kind of the studies and their quality scores (Section 3.2.3.3), and the availability of the artifacts produced as part of these studies (Section 3.2.3.2).

3.2.3.1 Projects analyzed

Figure 8 (a) shows information about the type of projects that were investigated in CI studies. We can observe that 40 of 101 studies (39.6%), analyze only open source projects. In contrast, 18 studies (17.8%) investigate private projects. On the other hand, 40 of the studies (39.6%) are not explicit about licenses of the projects, and 3 (3%) studies analyze mixed project settings, i.e., both open source and private projects.

Figure 8 (b) shows that 71 out of 101 studies (70.3%) do not investigate projects from a specific domain. On the other hand, 30 studies (29.7%) investigate domain-specific projects. For studies investigating specific project domains, we catalog 17 different domains (see Table 14). The most frequent domains are transports (4 occurrences), embedded systems (4 occurrences), telecommunications (3 occurrences), and software development (3 occurrences).

60 out of 101 studies focus on analyzing the historical data of software projects (e.g., production code or tests). The other remaining studies conducted interviews, surveys, or analyzed other units of information different from projects' source code, e.g., builds or companies. Figure 9 shows the descriptive statistics of the projects that were analyzed per study. We hide outliers for readability purposes—the highest outlier has 34,544 projects. While the mean of analyzed projects is 1,493 projects, the median is just 40, with a high

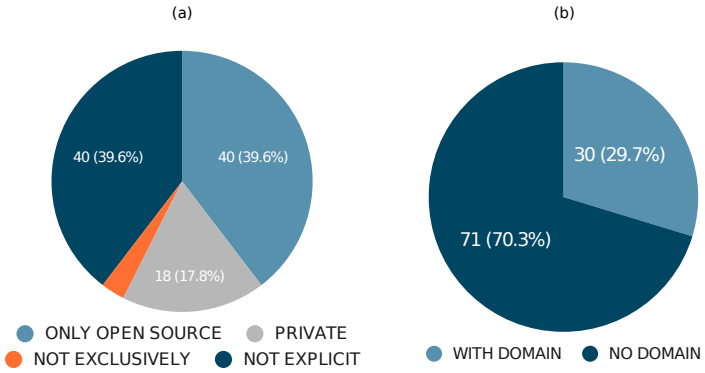


Figure 8 – (a) Proportion of studies based on the type of projects they analyze; (b) Proportion of studies that analyzed projects from specific domains (and vice versa);

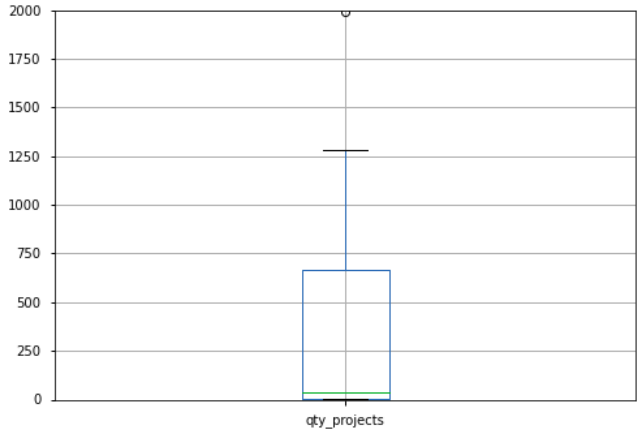
Table 14 – Applications domains investigated in primary studies.

Domain	Ocurrences
Transports	4
Embedded systems	4
Telecommuntications	3
Software Development	3
Web Application	2
Finance	2
Cloud Computing	2
Military Systems	2
Home and office solutions	1
Bookmaking company	1
Mobile software and social networks	1
Health care	1
HPC environment	1
Serverless applications	1
Neuroinformatics	1
Databases migration	1

frequency of studies analyzing just 1 or 2 projects. Some studies seem to be outliers, such as P72, which investigated 13,590 projects, and P100, which investigated 34,544 projects.

The P100 study uses a large corpus of projects (i.e., 34,544 projects) for specific investigations. For example, a large corpus of projects is used to identify which CI services are mostly used. However, to perform more specific investigations, P100 uses only a subset of the total corpus of projects (i.e., 1,000 projects). P72 highlights the TravisTorrent (BELLER; GOUSIOS; ZAIDMAN, 2017) dataset, which is a widely known dataset of projects from GitHub that collates build logs from TravisCI.

With the presented data, we can observe that, in general, the studies are distributed in various contexts, with the majority without a specific domain. Despite that, considering the 29.7% of the studies investigate domain-specific projects and the variance of CI among



Qty Projects	count	mean	std	min	0.25	0.5	0.75	max
	60	1,493.23	4,980.90	1	2	40	666	34,544

Figure 9 – Boxplot and descriptive statistics of the projects that were analyzed by our primary studies.

different domains and implementations (STåHL; BOSCH, 2014b; VIGGIATO et al., 2019), we suggest that the community employ studies that explore the differences in the CI implementations among such domains and how it influences the CI outcomes. We still draw attention to the high frequency of studies analyzing low-size samples.

3.2.3.2 Availability of Artifacts

Robles (ROBLES, 2010) investigated the MSR conference papers from 2004 to 2009. He found that the majority of published papers are hard to replicate. For example, although 64 out of 171 papers (37.4%) are based on publicly available datasets, these datasets are in the *"raw"* form and the papers do not provide the processed version of the datasets nor the tools that were used to process these datasets. Another 18.12% of papers (i.e., 31 papers) do not even provide the *"raw"* data to begin with.

Rodríguez-Pérez et al. (RODRIGUEZ-PéREZ; ROBLES; GONZáLEZ-BARAHONA, 2018) capitalize on the same issue of data availability and raise the concern about the reliability of the results from studies that are not reproducible studies. On the other hand, they (RODRIGUEZ-PéREZ; ROBLES; GONZáLEZ-BARAHONA, 2018) also report the increasing attention that the community has given to the issue of data availability over the last years. Therefore, in our study, we collect information about the availability of the artifacts used or produced in the primary studies, considering those that analyze projects. Figure 10 (a) reveals that 29 studies out of 60 (48.33%) provide publicly available datasets, while 31 studies (51.66%) do not provide publicly available datasets. From the studies providing publicly available datasets, all of them are studies using open source projects. On the other hand, those studies investigating private projects do not present dataset nor

anonymized nor in a raw manner, while some of the studies with mixed—both private and open source—projects provide only partial data referring to OSS projects. Other studies are not explicit regarding whether the dataset comes from private or OSS projects and does not present it.

Figure 10 (b) shows an increasing trend of pushing for data transparency in CI studies. Over recent years, we observe that the proportion of studies providing a publicly available dataset is higher than 50% (we consider only studies that analyze projects data). This increase in transparency might be due to initiatives from prominent conferences, such as the *artifact tracks*, in which authors are provided with special *badges* as a credit for their effort invested in sharing their artifacts. Nowadays, there are even awards to encourage the sharing of reproducible artifacts [3].

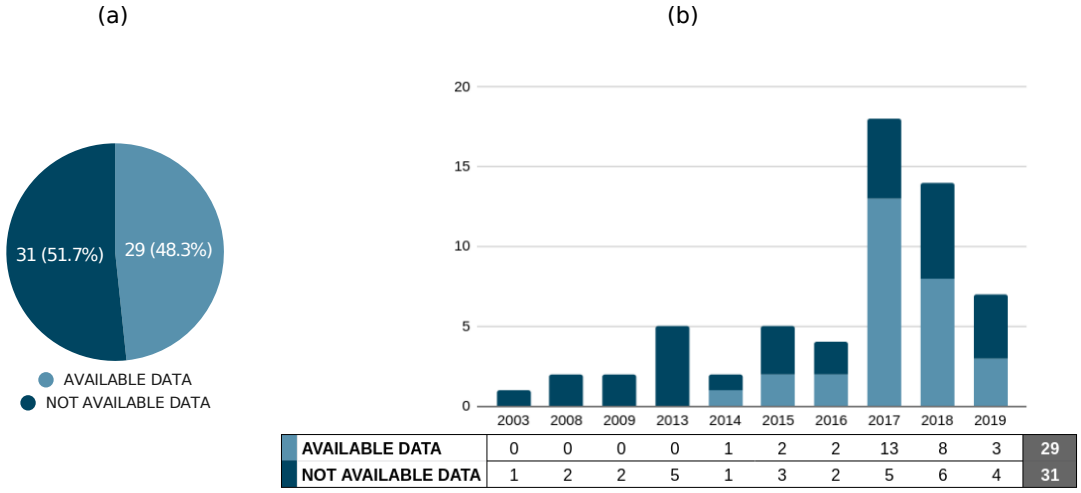


Figure 10 – (a) Proportion of studies according to data availability; (b) Proportion of transparency over the years.

3.2.3.3 Study Quality and Methodologies

Using the quality checklist and procedures presented in Section3.1.5, we assessed the methodologies of primary studies from which we extracted the findings discussed in Section 3.2.2 (38 papers). Similar to the approach applied by Dybå & Dingsøyr (DYBå; TORGEIR, 2008), such questions allow us to measure the reliability of the findings using the *quality score* as a proxy. By answering each question in the checklist (see Table 4) with a 1 (yes) or 0 (no), the sum of these values produces a quality score, as shown in the last column of Table 16. The remaining columns show the answers to each question of the checklist, while the rows represent each paper with claims discussed in this work.

As peer-review is an exclusion criterion, all of selected papers obtain 1 to Q1. The values for Q2 indicate whether studies have clearly defined aims. While the entire set of

[3] <https://icsme2020.github.io/cfp/ArtifactROSETrackCFP.html>

responses to Q1 through Q4 shows that the papers have, in general, a good report quality, in terms of rigour (Q5 to Q8), the obtained scores are lower, especially regarding Q8 - "Is there a comparison or control group?". Q8 has the lowest rate among primary studies with only five papers (P4, P14, P49, P50, and P79) having applied control groups to compare their results and findings.

The few studies with data available (12 out of the 38) impacts the credibility assessed by questions Q9 and Q10, i.e., "Does the empirical data and results support the findings?", respectively. The relevance (as expressed by Q11) of 30 out of 38 studies is clear and well discussed in terms of contributions to research and practice.

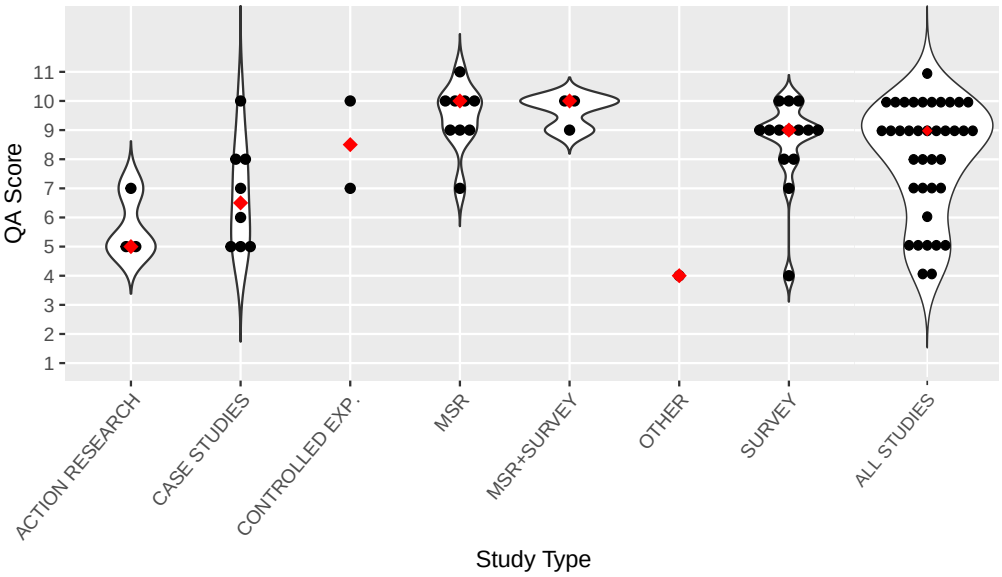


Figure 11 – Quality assessment scores per study type.

Figure 11 shows the scores grouped by type of study. The categories are (i) *mining software repository* (MSR) studies, and the other four classical empirical methods for software engineering: (ii) *controlled experiment*, (iii) *case study*, (iv) *survey*, and (v) *action research* (EASTERBROOK et al., 2008). We consider claims extracted from studies with higher scores to be more reliable than claims from studies with lower scores. We observe that MSR studies have higher quality scores, with a median of 10 points. The mixed-methods studies (MSR and survey) have the same median also with high scores (and even less variation). The last plot in Figure 11 shows the overall quality of the studies, with 4 as minimal score and a maximum of 11. The median score is 9.

Table 15 shows the average performance of the studies grouped by study type and the four quality criteria within our quality assessment checklist (see section 3.1.5). All types of studies performed well in quality of reporting, bordering the maximum score. In rigor, on the other hand, studies from action research and case studies have, on average, less than 2 points. This lower performance is significantly affected by question 6 regarding the metrics or concepts considered in the studies. Question 8, which concerns control or

comparison groups, also contributes to the lower performance in quality scores.

We suggest the software engineering community to strengthen the metrics and concepts and to employ comparison strategies. For example, when studying productivity, the studies should clarify the definitions of productivity (e.g., developer perception, story points, worked hours), how productivity is measured (e.g., questionnaires, issue trackers, management systems, work time), and the rationale behind this concept definition and measurement.

In the credibility criterion, case study and action research studies performed poorly, mainly in questions Q9 and Q10, which are regarding the evidence supporting findings and data availability. Lastly, regarding the relevance criterion (Q11), the selected controlled experiments scored low. Considering the results of the relevance criterion, we also recommend special attention to the development of case study and action research studies to be more transparent regarding data and to provide the rationale behind their conclusions. For example, although P50 performed a high-quality case study, P50 did not share its data for verification or reproducibility.

Table 15 – Quality Assessment per Kind of Study

	Quality of Reporting (0..3)	**Rigor (0..4)**	**Credibility (0..3)**	**Relevance (0..1)**
ACTION RESEARCH	2.66	1.33	1.00	0.66
SURVEY	3.00	2.50	2.14	0.92
CONTROLLED EXP.	3.00	2.50	2.50	0.50
CASE STUDIES	2.75	1.87	1.50	0.62
MSR	2.88	3.00	2.66	0.88
MSR+SURVEY	3.00	3.33	2.66	0.66

Figure 12 shows the proportion of claims per study type, revealing that survey research was the method applied the most concerning the extracted claims (47.2% - 59 claims)—followed by case studies (19.2% - 24 claims) and MSR (17.6% - 22 claims). Only 7 (5.6%) of the claims are associated with a mixed-methods approach, emerging from 3 studies (P4, P7, and P89). The largest proportion of the claims — 70.4% emerges from MSR, Surveys, and mixed-methods studies, which is a promising given that such categories of studies obtain the highest quality scores.

In Figure 13, we analyze the occurrence of each study type within the studied themes. We observe that the *Integration Patterns* theme has 14 out of 22 claims (63.64%) made by mining software repositories (MSR) or mixed-methods (which are categories with the highest quality scores).

The *Issues & Defects* theme also has a significant frequency (6 out of 14 - 42.86%) of claims from MSR and mixed-methods. With respect to Quality Assurance, the theme has 23 claims, but there is a huge concentration (73.91%) of claims made from surveys, which can suggest the need for more complementary study types.

Table 16 – Quality Assessment.

Study	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**	**Q9**	**Q10**	**Q11**	**Total**
P4	1	1	1	1	1	1	1	1	1	0	1	**10**
P6	1	1	1	1	0	0	0	0	0	0	1	**5**
P7	1	1	1	1	1	1	1	0	1	1	0	**9**
P9	1	1	1	1	0	1	1	0	1	0	1	**8**
P14	1	1	1	1	1	1	1	1	1	1	1	**11**
P24	1	1	1	1	0	1	1	0	1	0	1	**8**
P25	1	1	1	1	1	1	1	0	1	0	1	**9**
P29	1	1	1	1	1	1	1	0	1	0	1	**9**
P31	1	1	1	0	0	0	1	0	0	0	1	**5**
P38	1	1	1	1	0	0	0	0	0	0	1	**5**
P39	1	1	1	1	1	0	1	0	0	0	1	**7**
P40	1	1	1	1	1	1	1	0	1	0	1	**9**
P44	1	1	1	1	1	1	1	0	1	0	1	**9**
P46	1	1	1	0	1	0	1	0	0	0	0	**5**
P47	1	1	1	1	1	1	1	0	1	1	1	**10**
P49	1	1	1	1	0	1	0	1	1	0	0	**7**
P50	1	1	1	1	1	1	1	1	1	0	1	**10**
P52	1	1	1	1	1	1	1	0	1	0	1	**9**
P58	1	1	1	1	0	0	0	0	0	0	0	**4**
P59	1	1	1	0	0	1	1	0	1	0	1	**7**
P64	1	1	1	1	0	0	1	0	0	0	0	**5**
P69	1	1	1	1	1	1	1	0	1	1	1	**10**
P73	1	1	1	1	1	1	1	0	1	0	1	**9**
P74	1	1	1	1	1	1	1	0	1	1	1	**10**
P79	1	1	1	1	1	1	0	1	1	0	0	**8**
P81	1	1	1	1	1	1	1	0	1	1	1	**10**
P89	1	1	1	1	1	1	1	0	1	1	1	**10**
P90	1	1	1	1	1	1	1	0	1	1	0	**9**
P91	1	1	1	1	0	1	1	0	1	0	1	**8**
P92	1	1	1	1	1	1	1	0	1	0	1	**9**
P93	1	1	0	0	0	0	1	0	0	0	1	**4**
P97	1	1	1	1	1	1	1	0	1	1	1	**10**
P99	1	1	1	1	1	1	1	0	1	0	1	**9**
P100	1	1	1	1	1	1	1	0	1	1	1	**10**
P102	1	1	1	1	1	0	1	0	0	0	0	**6**
P104	1	1	1	1	1	1	1	0	1	1	1	**10**
P105	1	1	1	1	1	1	1	0	1	0	1	**9**
P106	1	1	1	1	1	0	0	0	0	1	1	**7**
Total	**38**	**38**	**37**	**34**	**27**	**28**	**32**	**5**	**28**	**12**	**30**	

Lastly, during the extraction phase (Section 3.1.4), we recorded methodological

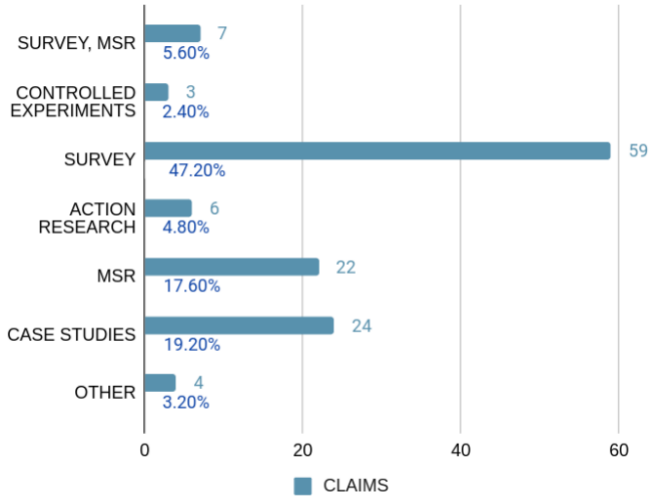


Figure 12 – Proportion and quantity of claims per study type;

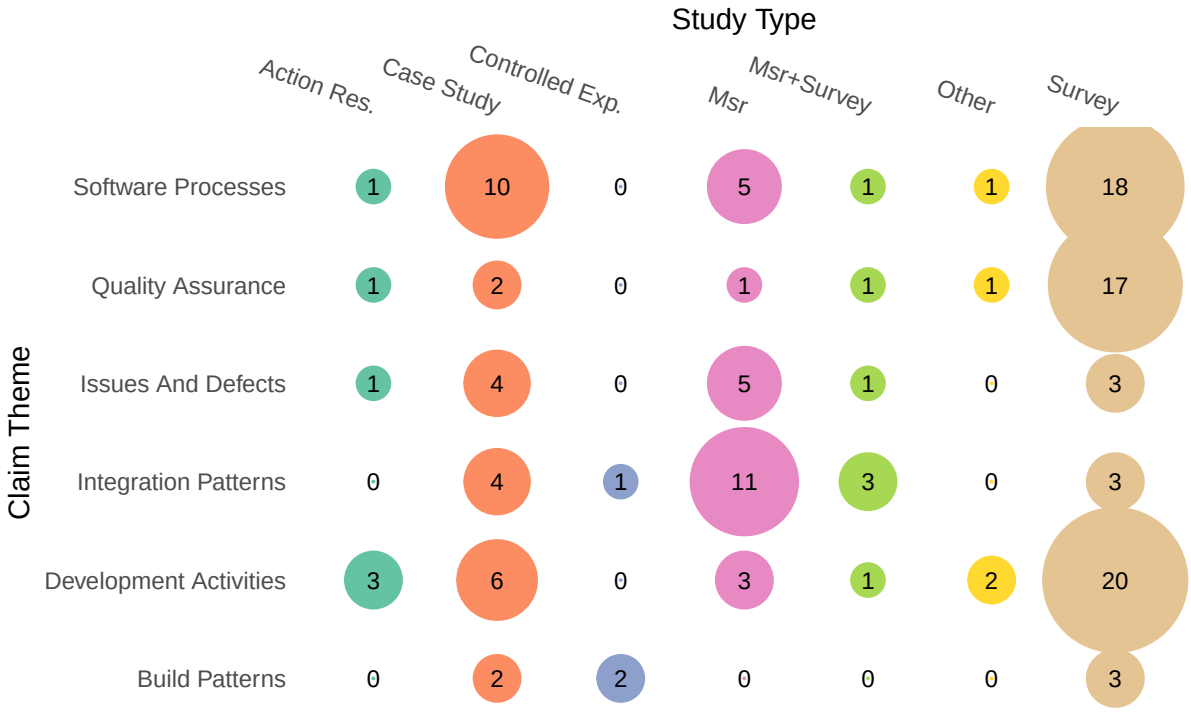


Figure 13 – Claims quantity for each theme and study type.

instruments used by the primary studies to confirm their findings. Table 17 reports a summary of statistical tests, models, and qualitative methodological instruments identified in the selected studies. We extract methods ranging from statistical tests, such as Cliff's delta, and Mann-Whitney-Wilcoxon, to qualitative methods such as thematic analysis and interviews.

Table 17 – Methodological instruments applied in the studies to confirm findings.

Instruments	Studies
ANOVA	P91, P104
Cliff's delta	P14,P25, P81, P90, P105
Cohorts comparison	P49
Cronbach alphas and Factor analysis	P9
Fisher's exact test	P100
Interview	P97, P99
Linear Regression and ANOVA	P91
Logistic Regression	P4, P74
K-Means	P14
Mann-Whitney-Wilcoxon test (MWW)	P14, P25, P81, P90, P100, P105
Mixed-effects RDD Model	P89
Multiple Linear Regression	P47, P104
Survey	P7, P52, P89, P97, P99
Survey Average Score and Standard Deviation	P52
Thematic Analysis	P24

3.3 Discussion

In the previous sections, we presented the findings from this SLR related to the research questions. In this section, we discuss the results, beginning with some methodological aspects. Then, we identify and discuss limitations on literature concerning the considered setup of continuous integration. Finally, we highlight some research opportunities.

3.3.1 CI Environment and Study Results

As discussed in Section 2.1, we adopt practices based on Duvall et al. (DUVALL, 2013) and Fowler (FOWLER; FOEMMEL, 2006) to identify implementation of CI. From these criteria, we find (see Section 3.2.1) that 42.5% (43) of the CI studies do not discuss or present any of these specific criteria. On the other hand, 15.8% (16) of studies apply one criterion. The results is an alarming proportion of 58.3% of primary studies having none or only one criterion to identify whether CI has been implemented in a project. This is alarming because this suggests that most of the existing claims regarding CI might be biased towards projects that do not consistently implement CI. The most frequent criterion specified for 45.5% of the studies is the usage of an online CI service (see Figure 5 (b)), which allows implementing a CI pipeline for existing projects. However, this finding represents a challenge to be overcome by the research community since other studies revealed that CI usage may be inconsistent, sporadical, or discontinued (VASILESCU et al., 2015).

Vasilescu et al. (VASILESCU et al., 2015) investigated CI quality and productivity outcomes. From a dataset of 918 GitHub projects that used TravisCI, they found that only 246 projects have a good level of activity using TravisCI, while the other 672

projects have used TravisCI only for a few months. Thus, it suggests that solely relying on CI service configuration is not enough to determine a proper CI adoption.

Vassalo et al. (VASSALLO et al., 2019) performed a survey with 124 professional developers confirming that deviations from CI best practices occur in practice and can be the cause of CI degradation. They mined 36 projects and verified relevant instances of four anti-patterns (DUVALL; OLSON, 2011): late merging, slow build, broken release branch, and skip failed test.

Felidre et al. (FELIDRé et al., 2019) also investigated CI bad practices, beyond slow build and broken release branch, they shed light on infrequent commits and poor test coverage. In addition, their analysis of 1,270 open source projects confirmed the existence of a phenomenon known by practitioners as *CI Theater*, which refers to self-proclaimed CI projects that do not really implement CI (THOUGHTWORKS, 2017).

Considering the findings observed in Section 3.2.1, and the studies mentioned above, we suspect that there are few studies considering a more robust number of criteria in order to perform a more rigorous evaluation of CI adoption. In line with Ståhl & Bosch (STåHL; BOSCH, 2014b), we observe that simply stating that projects use continuous integration is not sufficient. There is an urgent need to classify which practices and at what level such projects implement them. It is especially true if we consider CI as a set of practices, where the benefits and challenges related to CI are directly related to the usage of such a set of practices.

3.3.2 Research Opportunities

Beyond the research opportunities already discussed in the themes of Section 3.2.2, this subsection discusses existing gaps in the research on continuous integration and apparent contradictions among the findings, especially focusing on the themes "integration patterns" and "development activities".

3.3.2.1 Integration Patterns

Regarding the integration patterns theme, we observe that CI may influence the processing of pull requests in different stages. Figure 14 shows the claims related to how CI influences the processing of pull-requests. Figure 14 also shows to which stage of the pull-request life-cycle a claim refers. P81 reveals evidence that projects tend to have more pull request (PR) submissions after they adopt CI. P47 and P74 state that CI influences PR acceptance, and CI projects tend to have more closed PRs. After merging, CI is a helpful tool to detect merging issues earlier (P69). Moreover, P81 found that CI projects deliver more PRs and more rapidly.

All of these claims represent the following codes: (i) "CI is related to a positive impact on pull request life-cycle" having seven studies supporting studies (P47, P69, P74,

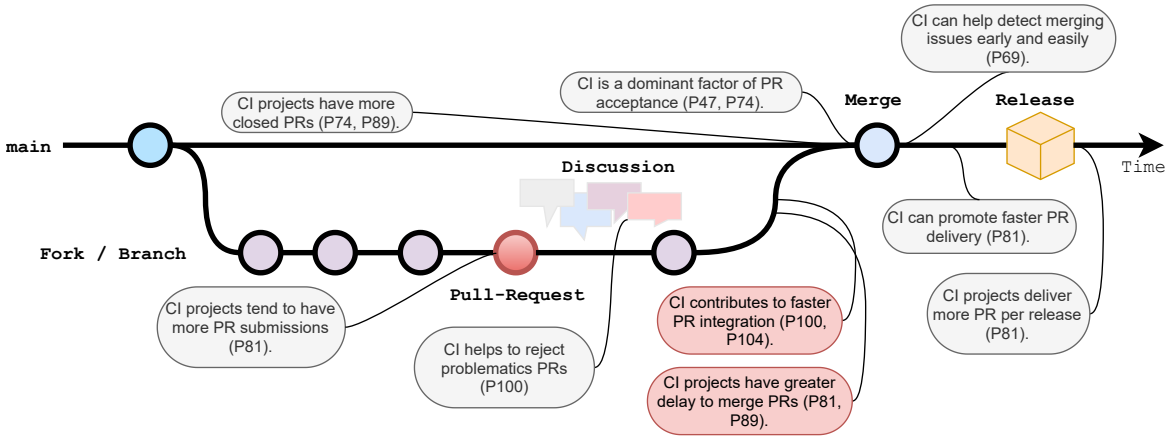


Figure 14 – Claims related to the effects of CI on pull requests life cycle.

P81, P89, P100, P104); and (ii) "CI is related to negative impact on pull requests life-cycle" with two studies supporting (P81, P89). There is an apparent contradiction regarding the time to integrate a pull request. Four studies investigate the time to merge pull requests and its relation with continuous integration — P104 from 2015, P100 from 2016, P81 from 2018, and P89 from 2017.

In 2015, P104 investigated 103,284 pull requests from 40 different projects using multiple linear regression models to evaluate the time to merge pull requests. P104 observed that CI shortens the time to merge pull requests. Later, in 2016, another MSR study (P100), including 1,529,291 builds and 653,404 pull requests, P100 observed that CI build statuses can influence the development team to merge pull requests more quickly.

On the other hand, P89, which studied the time to merge pull requests using an Regression Discontinuity Design (RDD) model on 77 projects, found that, on average, pull requests have a trend to take a longer to be merged as the project matures, with CI having no apparent impact in this trend, i.e., CI projects keep increasing the time to merge PRs regardless of their adoption of CI. Finally, in 2018, P81 analyzed 87 projects and concluded that projects may take longer to merge pull requests after adopting CI. The difference is small but statistically significant.

This apparent contradiction between P81 and P89 against P100 and P104 might be related to several factors, including (as observed earlier) how these studies determine whether projects are using CI or not. While P81 and P104 consider the start of CI adoption as the moment when the first automated build is created in a CI Service, P100 does not identify a certain moment to identify when CI has been adopted. Instead, P100 grouped pull requests into two categories: with and without build information from the CI server. We conjecture that the main factors contributing to such apparent contradiction might be: (i) the age of projects influencing the longevity of pull requests according to the above-mentioned P89 findings (i.e., the older a project is, the longer the merge delay); (ii) The P81 finding regarding an increasing trend in PR submissions after CI adoption

may explain why there is an extra time to process pull requests. Therefore, we argue that more studies are necessary to investigate the claims related to CI and pull-request lifetime considering these possible confounding factors, such as project age, the number of opened PRs, among others.

3.3.2.2 Development Activities

From the "Development Activities" theme, we find opportunities to construct a deep understanding of some reported phenomena, such as what we highlight as "confidence contradicting claims" and "productivity contradicting claims". The "confidence contradicting claims" is marked by studies that make claims related to the code "CI may generate a false sense of confidence", while some studies raise claims under the code "CI is associated with confidence improvement". The "productivity contradicting claims" refers to studies claiming that "CI is related to an increase in productivity or efficiency", while some studies claim that "CI is associated with a decreased perception of productivity".

Confidence contradicting claims. Table 18 shows the claims related to developer confidence. Six studies provide support to conclude that CI improves developer confidence (P39, P97, P100, P106, P58, P93). On the other hand, two studies claim that CI can promote a false sense of confidence (P58, P106).

Table 18 – Claims related to the effects of CI developer confidence.

CODE: CI is associated with confidence improvement	
Claim	**Studies**
CI increases the confidence about the quality.	P39
CI makes the team less worried about breaking build.	P97, P100
CI improves the developers confidence to perform the required code changes.	P106, P58, P93
CODE: CI may generate a false sense of confidence	
Claim	**Studies**
The false sense of confidence is a recurring problem in CI.	P58, P106
Flaky tests may challenge CI projects.	P106

P93 theorizes that CI allows programmers to assume themselves as single-programmers in a project, supporting an improvement in confidence. For instance, by relying on the lower number of new inconsistencies expected in each integration cycle, the developer can behave as if they were the only person modifying the code, reducing the cognitive tractability of programming. In the same line, P58 and P106 surveyed 158 CI users and reported the perception of respondents that CI provides more confidence to perform the required code changes.

Other studies may help to understand this boost in confidence better. Developers seem to delegate quality assurance to CI service and rely on its feedback. P39, an experience

report, sheds light on improved confidence in product quality after CI adoption due to test automatization. P100 reports a survey with 407 respondents and reveals that the most common reason to use CI is the expectation that it makes developers less worried about breaking builds. After a triangulation between an interview and two surveys, P97 reports the same finding.

On the other hand, P58 and P106 also shed light on a reported problem of a false sense of confidence. This situation occurs when developers rely on an environment that may suffer from low quality or insufficient tests. The lack of balance between developer trust and the environment's trustworthiness determines the occurrence or not of overconfidence. The environment may provide a baseless trust and suggest an opportunity for practitioners and researchers to investigate and supply developers with objective criteria and guidance to define a reliable CI environment to avoid the mentioned false sense of confidence. For example, what minimum set of practices or metric values should we achieve before having a reliable CI environment and feedback that can be trusted?

Additionally, these studies raise substantially different aspects of confidence: (i) confidence in the product quality (P39); (ii) personal confidence to perform tasks (P106, P58, P93); and (iii) confidence in the process reliability (P97, P100, P58, P106). Nonetheless, none of the mentioned studies addressed the confidence question directly, and therefore did not provide a theoretical base to analyze confidence. The studies, in general, registered developers' perceptions, leaving room to further investigation and theory formulation about developer's confidence and the role of CI.

Productivity contradicting claims. Table 19 shows the claims related to development productivity. There are 12 claims in 11 studies supporting the code "CI is related to productivity and efficiency increase", and one study claiming that "CI is associated with a decreased perception of productivity".

P52 performs a case study with four projects and validates the hypothesis that CI contributes to an increase in the developer productivity due to parallel development and reducing tasks before checking in (i.e., committing). Through another case study, P59 confirms this claim, while P39 and P31 share different experience reports that record an increase in development efficiency and throughput per developer, respectively.

P73 reports interviews, and P97 presents a triangulation between an interview and two surveys. They both confirm the perceptions that CI increases productivity. By mining software repositories from 246 projects, P74 finds that external contributors tend to have fewer pull requests rejected if CI is adopted. Other studies such as P100, P102, and P106 also bring results corroborating this code.

In opposition to these studies and findings, P91 investigates the links between agile practices, interpersonal conflict, and perceived productivity. P91 presents a survey with 68 software developers. P91 grounds its research method in the Integrated Model of Group Development (IMGD) — a theory on group development (WHEELAN; HOCHBERGER,

Table 19 – Claims related to the effects of CI on development productivity.

CODE: CI is related to productivity and efficiency increasing	
Claim	**Studies**
CI increases development efficiency (due to the automation of tasks and fast feed back)	P39, P52, P73, P59, P31, P97
CI is associated with external contributors having fewer pull requests rejected.	P74
CI decreases the debug time.	P100, P79
CI allows quickly grow of source code.	P102
CI speed up development practice.	P106
CI Reduced integration problems allowing the team to deliver software more rapidly	P79
CODE: CI is associated with a decreased perception of productivity	
Claim	**Studies**
The CI adoption leads to a worsening in the perceived team productivity.	P91

1996) and a tool (questionnaire) to employ psychological measurement of the stage where a group is in a developmental perspective. Moreover, P91 applies two other surveys to measure agile practices and the perceived productivity. It concludes:

> *"I have also shown that with higher scores on Continuous Integration and Testing came lower scores on this perceived productivity measurement. That means that the more continuous integration and testing the team conducts, the worse is the perceived team productivity. However, I do not have any external measurement of the productivity of the teams and can not draw conclusions on the actual productivity [...]".* (p. 4)

While P91 get productivity as developer perceives (an internal measurement), other studies quantify productivity by the time spent, e.g., adding features vs. debugging (P73, P79, P100), time saved (P52, P59, P106), integrator productivity to merge pull requests (P74), others by developer throughput (P31, P102). Thus, we can suppose that the measurement strategy of productivity could explain the difference in the findings from P91. However, P97 also registers developers' perceptions and finds a positive influence of CI.

That way, considering that 11 studies are going in a direction claiming that "CI is related to productivity and efficiency increasing" and only one study declaring a worsening in the perceived productivity, we are led to consider the participants of these studies. While P91 surveys 68 software developers from three big companies (a telecommunications equipment and services company, an aerospace and defense company, and an automotive parts manufacturing company), P97 surveyed 574 developers (51 from one software engineering solutions company and 523 from a broad group on the internet).

Such a difference in findings may be due to the smaller number of participants in P91, or the difference in the domains of their companies as well as their CI practices (STåHL; BOSCH, 2014b; ZHAO et al., 2017; VIGGIATO et al., 2019). Nevertheless, as mentioned in P91, it is essential to point out that the perceived productivity may be affected by subjective factors, such as the kind of work performed by the developer. For example, code review may be necessary from an organizational perspective but can be seen as not as productive by a particular developer, decreasing their perceived productivity.

3.4 Threats to validity

The goal of our SLR is to provide a summary of the effects of CI on the software development phenomena. We follow the guidelines provided by Kichenham & Charters (KEELE et al., 2007) to develop our review protocol while defining strategies to mitigate possible bias. However, as it happens to every study, our SLR is not without flaws and, in this section, we discuss the limitations of our study.

3.4.1 Search Strategy

The search strategy may have bias or limitations on its search string and expression power, the limitations on search engines, and publication bias, i.e., positive results are more likely to be published than negative (KEELE et al., 2007). To mitigate the search string threats, we apply several identified synonyms to reach the effects of continuous integration, the intervention studied. In addition, aiming to reduce the limitations of search engines, we use six different search engines including five formal databases and one index engine, following the recommendations from Chen et al. (LERO, 2010), thereby including journals and conferences publications—which contributes to publication bias mitigation.

3.4.2 Screening Papers

The screening and selection phase (see Section 3.1.3.2) follows the inclusion and exclusion criteria defined during the protocol definition, as recommended by Kitchenham & Charters (KEELE et al., 2007) to mitigate the selection bias. In addition, the decision relied on the evaluation of two researchers and the agreement was measured using the Cohen Kappa statistic (COHEN, 1968). A substantial agreement was achieved both in the first screening (0.72) and in the snowballing phase (0.76). The disagreements were read and arbitrated by a third researcher.

3.4.3 Data Extraction

To reduce the possibility of bias in the data extraction phase, we proceed the following steps (see Section 3.1.4) to mitigate it. First, the meta-data was retrieved in an automated process using the data obtained in Mendeley - the reference management tool adopted. To avoid mistakes or missing information, the processed meta-data was manually inspected by one researcher. Second, the definition of extraction form (see Table 3) was available in the review protocol and in a web host to all three readers. Third, to decrease the chances of inattention, lack of understanding, or any other reason for mistaken data collection, the reading of each paper was performed by two researchers that filled the extraction form independently. Fourth, to treat the disagreements in the extraction and also avoid bias, each pair discussed the extracted data to achieve a consensus.

3.4.4 Quality Assessment

The quality assessment stage was performed based on a quality checklist composed of eleven questions. The threats in this phase can potentially reflect on data extraction and data synthesis in such a way that (i) the researchers may need to comprehend the questions better or (ii) the questions may not sufficiently express the quality of the papers. To mitigate this, we: (i) developed a questionnaire inspired by the previous experiences reported by Dybå and Dingsøyr (DYBå; TORGEIR, 2008) and Kitchenham and Charters (KEELE et al., 2007); (ii) the checklist covers four distinct quality aspects (quality of reporting, rigour, credibility, and relevance); (iii) we ran two rounds with pilot papers with three researchers together to assess the understanding of the quality checklist; then we (iv) provide the most of questions with instructions, i.e., minor questions to support their assessment and answer.

3.4.5 Data Synthesis

As described in Section 3.1.6, our study explores quantitative (RQ1 and RQ3) and qualitative synthesis (RQ2). In the quantitative synthesis of RQ1 and RQ3, we present a summarization to create a landscape of the studies and point out some directions to researchers. The main threat in these syntheses is related to the chosen criteria.

First, in RQ1, while investigating how primary studies identify or classify their subjects as a CI project, we found no clear definition of which practices determine whether a given project uses CI or not. Studies revealed that there are many variants of CI implementation (STåHL; BOSCH, 2014b; ZHAO et al., 2017; VIGGIATO et al., 2019). Therefore, our chosen criteria to identify whether a project uses CI or not may not perfectly match CI usage for every context. Nevertheless, we decided to use the prescriptive list of practices from Duvall et al. (DUVALL, 2013), and Fowler (FOWLER; FOEMMEL, 2006) because they have been the most used definition of CI in existing research so far

and present a prescriptive list of practices. In this way, we adopted a set of seven generic practices inspired in their lists. Second, in the RQ3, we analyze the studies quality and methodologies relying on our data extraction and quality assessment, then subject to risks presented in Sections 3.4.3 and 3.4.4.

In the qualitative analysis of RQ2, we follow the guidelines of Cruzes & Dybå (CRUZES; DYBå, 2011) to perform a thematic synthesis. In the coding phase, to mitigate the threats of confirmation bias, we first use an inductive approach performed by two researchers to define the set of codes. Second, to avoid a wrong grouping, two researchers coded all the extracted segments independently (this step also achieved a substantial Kappa agreement rate - 0.73), and a third researcher resolves the disagreements. Finally, all the authors discuss and agree with the translation of the codes into the presented themes.

In Sections 3.2.2 and 3.3.2, we assess the trustworthiness of the synthesis in terms of type of studies, number of occurrences, and the relationship between the findings of different primary studies.

3.5 Conclusion

We perform a systematic literature review (SLR) on the effects of continuous integration on the software development phenomena. Our main goal is to summarise the existing empirical evidence and body of knowledge regarding CI to support a better decision process, avoiding overestimating or underestimating the results and costs of CI adoption. We collect and analyze empirical evidence from 101 primary studies ranging from 2003 to 2019, conducting quantitative and qualitative analyses. We hope our study can support an evidence-based practice by development teams and organizations to build work policies. Our study can also serve as a map regarding which claims related to CI should be more thoroughly studied in the future (i.e., given the rigour of the state-of-art studies).

3.5.1 Results and Implications

The collation of findings related to the effects of CI and their accompanying evidence (see Sections 3.2.1, 3.2.2, 3.2.3, and 4.3) can be useful for researchers and practitioners. In Sections 3.2.1 we show that 42.5% of the primary studies did not present explicit criteria to identify projects that use CI. We also found that 15.8% used only one criterion (e.g., more than half of studies used "automated builds" as a criterion). This finding reveals a weakness in our current empirical literature since identifying whether a project uses CI or not is at the core of how we analyze the effects (positive or negative) of using CI. As an implication, we believe that there are plenty of research opportunities to re-evaluate existing analyses

by using more robust criteria to identify CI projects. For example, checking whether they use automated builds and also how frequently they perform commits.

Regarding the criteria applied to check whether participants use CI or not (Sections 3.2.1 and 3.3.1), our findings reveal the need for performing other checks during interviews or surveys related to the adherence of CI beyond the self-declaration, e.g., *"on a scale of 1 to 7, how would you classify that your project adheres to CI?"*. Studies may consider, for example, checking which practices the subjects use in their CI environment before classifying them as CI projects.

Sections 3.2.2.1 and 3.3.2.2 discuss our findings related to the effects of CI on *development activities*. We find evidence for the association between CI and improved productivity, efficiency, and developer confidence. On the other hand, other findings suggest that CI may introduce complexity to the project, requiring more effort and discipline from developers, negatively impacting developers' perceived productivity. Some studies also discuss the false sense of confidence, i.e., when developers blindly rely on flaky tests.

Continuous integration benefits the *software process* (see Section 3.2.2.2) by promoting faster iterations, more stability, predictability, and transparency in the development process. Although CI may incur technical challenges to the team (e.g., creating a reliable automated build environment), CI is mentioned as a success factor in software projects. Regarding *quality assurance*, Section 3.2.2.3 reveals evidence on the association between CI and better testing. The studies demonstrate a perceived provision of transparency and continuous quality inspections when CI is adopted.

With respect to *integration patterns* (Section 3.2.2.4) our study indicates that CI positively influences the way developers perform commits (e.g., increasing the frequency and decreasing the size of commits). CI can also benefit the pull-based development by improving and accelerating the integration process. However, there are also studies reporting that CI may prolong the pull request lifetime. Section 3.3.2.1 discusses in detail the way CI impacts differently in each stage of the pull-request life-cycle.

Regarding *issues & defects* (Section 3.2.2.5), we find that studies credit CI to an improvement in the time to find and fix issues. They also report a decrease in defects reported. About *build patterns* the studies reveal that CI impacts the build process (Section 3.2.2.6), promoting good practices related to build health and contributing to an increase in successful builds.

Lastly, regarding RQ3, Section 3.2.3.1 shows that there is a wide variety in the primary studies (in terms of domains and subjects). The number of studies making their datasets available is growing over the past few years (Section 3.2.3.2). The studies from which we extract claims (38 out of 101) have a notable overall quality (median score of 9 out of 11 — $\tilde{Qscore} = 9$), mainly those that use MSR and mixed-methods as their methodologies, both with $\tilde{Qscore} = 10$, while survey researches obtain a $\tilde{Qscore} = 9$. Most of the claims (70.4%) emerge from these three study types.

3.5.2 Open questions for Practitioners and Researchers

Given our observed results, we believe that continuous integration has plenty of room for future empirical studies and new tools to address open questions or strengthen the current body of knowledge. Section 3.2.1, for example, shows that a community effort to build a solid foundation about how to classify projects using CI could be useful to further empirical studies, e.g., a "CI maturity score" could be conceived, or a consensual set of minimal practices could be established by researchers and practitioners.

Sections 3.2.2.1 and 3.3.2.2 highlight that researchers should stay attentive to how factors such as productivity and confidence are measured since there is significant diversity among primary studies. For example, some studies assess the developer perception of productivity, while others consider the time to merge a pull request. Other examples are studies assessing developers' confidence, in which some investigate the developers' confidence in performing certain tasks, while others analyze confidence in terms of trusting CI. The *development activities* theme reveals that there is a need for guidelines and metrics to inform practitioners about the reliability of their CI environment, avoiding the false sense of confidence phenomenon. This phenomenon has a link with the quality of the tests and their consequent reliability. In Section 3.2.2.3, regarding *quality assurance*, we find evidence for the association between CI and an increase in volume and coverage of tests. However, more studies are necessary to understand the relationship between test effort and test quality in CI.

Section 3.2.2.2 discusses open challenges to practitioners, researchers, and tool builders. Multiple studies report the difficulty faced by developers with the technical activities, such as configuring the build environment. Practitioners and tool builders may consider such challenges and elaborate strategies and tools to mitigate them. We propose that further studies are necessary to better understand the trade-offs between adopting CI and overcoming its inherent challenges (e.g., trade-offs between automation, technical challenges, and perceived productivity as discussed in Sections 3.2.2.1, 3.2.2.2, and 3.3.2.2).

The results of RQ3 show that 29.7% of the included studies investigate domain-specific projects (section 3.2.3.1), which highlights the need for studying whether CI is better adopted in certain domains (e.g., web application, embedded systems, finance, among others) (STåHL; BOSCH, 2014b; VIGGIATO et al., 2019). Section 3.2.3.3 reveals that researchers should be aware of the low amount of studies applying comparison or control groups to assess their findings and suggests that more diverse and complementary studies may be necessary for *quality assurance*. For example, MSR studies assessing the evolution of the test code in project repositories.

4 Continuous Integration and Software Quality: A Causal Explanatory Study

An earlier version of this chapter is under review in the Empirical Software Engineering.

Our study investigates potential causal relationships between continuous integration (CI) and software quality. This study is important because understanding causal effects can help practitioners to measure the actual benefits of CI. Given the predominance of statistical correlation studies, this work extends the current CI knowledge by offering causal conclusions. Our study helps researchers to remain aware of possible confounders, related variables, and adjacent associations when investigating the relationship between CI and software quality.

Through a systematic literature review (SLR), Soares et al. (SOARES et al., 2022) investigated the influence of CI on software development. Their study compiles most of the existing (non-causal) associations between CI and software quality. Based on current findings, the SLR concludes that CI may improve the time to develop and merge addressed issues and reduce the number of reported defects (ZHAO et al., 2017; RAHMAN et al., 2018; SOARES et al., 2022). However, the SLR also identifies several CI associations that need further investigation. For example, the relationship between the development environment reliability and developers' (over)confidence is due to trusting CI outcomes.

Despite all the effort invested in studying the potential benefits of CI (SOARES et al., 2022; STåHL; BOSCH, 2014a; ZHAO et al., 2017; VASILESCU et al., 2015; KAYNAK; ÇILDEN; AYDIN, 2019; PINTO et al., 2018; CASSEE; VASILESCU; SEREBRENIK, 2020), the software engineering community still needs to benefit from a step further, which is to investigate causal relationships in existing studies. The difference between association and causation is critical. For example, inferring causal conclusions from associations may be harmful because of spurious associations, leading to false conclusions (e.g., confounding effect) (PEARL et al., 2000; GREENLAND; PEARL; ROBINS, 1999b). Unconsidered relationships among variables may confound the causal assumptions about the studied phenomenon. For example, one can observe that streets have puddles and people wear raincoats whenever it rains. However, if one assumes that puddles cause people to wear raincoats, one would fail to consider a relationship of a third variable (*rain*) that causes both puddles and people wearing raincoats.

Nevertheless, despite the famous adage states that "correlation does not imply causation" (i.e., statistical associations are not sufficient to determine causal relationships), it is also true that "there is no causation without association." Reichenbach formally links statistical association with causal structures (PENROSE; PERCIVAL, 1962; PETERS; JANZING; SCHöLKOPF, 2017) (see Section 2.3). Therefore, it is still possible to infer the existence of causal links from statistical dependencies (i.e., functional relationships between the variables) (PETERS; JANZING; SCHöLKOPF, 2017). To infer causation, Pearl (PEARL et al., 2000) proposed employing a causal modeling framework based on Causal Directed Acyclic Graphs (causal DAGs). In section 2.4, we explained the theory proposed by Pearl because we apply his proposed theory in our study.

To study the causal relationship between CI and software quality, we use an approach that consists of five interconnected stages. Each stage is guided by a research question described in the following.

RQ1. What does the literature proclaim about CI and software quality? To understand the variables that can potentially play a role in the relationship between CI and software quality, we conduct a literature review, which helps us define a causal DAG (i.e., a graphical-statistical technique - to enable us to draw domain assumptions and infer causal conclusions in a later stage (HERNáN; ROBINS, 2010)). The goal of the review is to discover the existing associations between CI and software quality, as well as marginal associations (e.g., associations between code smells and software quality), which will help us to draw a comprehensive DAG containing the existing studies' assumptions about how CI may influence software quality (and the potential confounding variables surrounding both CI and software quality). To study this relationship, we consider bug reports a proxy for software quality, similar to a previous study from Vasilescu et al. (VASILESCU et al., 2015).

RQ2. Is the causal effect of CI on software quality empirically observable? Once we have a causal DAG containing a sufficient set of variables to analyze the relationship between *ContinuousIntegration* and *BugReport*, we can apply the d-Separation rules (PEARL; JUDEA, 1994) and evaluate the raised set of testable statistical implications from the DAG built in RQ1. d-Separation is a set of graphical rules to identify if an association path exists between two or more variables in the DAG. From such associations, testable implications arise (d-Separation will be explained in Section 2.4.1). Next, we mine software repositories to collect observational data on the variables of the DAG and perform (un)conditional independence tests on our dataset to answer RQ2. Note that this differs from an ordinal "causation is not correlation" MSR study, as these tests are guided by the d-Separation rules derived from the causal DAG.

RQ3. What would be an accurate causal theory for CI? Considering the investigations in RQ1 and RQ2 and the testable implications from the causal DAG in the dataset, we can analyze the hypotheses that failed in our analyses (i.e., statistical

implications from the supposed relationships between the variables that were not supported by the data) and propose a new causal DAG. We do so, again, using the literature knowledge and empirical data to answer RQ3 but now considering the new and corrected causal DAG this time.

4.1 RESEARCH METHOD

We follow a pipeline of 5 stages shown in Fig. 15 to answer our research questions: The Literature Review (stage 1) and DAG Building (stage 2) stages contribute to answering RQ1. The data collection from repositories (stage 3) and the DAG Implications Testing (stage 4) stages address RQ2. Lastly, another stage of DAG Building (stage 5) performed in an iterative manner with the DAG Implications Testing stage (stage 4) until a final DAG is obtained containing literature and data consistency, answers RQ3. Next, we detail the different stages of our research methodology.

As discussed in section 2.4, prior knowledge is an existing approach to build causal structures regarding a phenomenon. Then in this first research question, we intend to draw a causal DAG based on the literature assumptions regarding continuous integration (CI) and software quality. Although most of the studies in software engineering are association studies, we discussed the relationship between association and causation in section 2.3 and how d-Separation rules raise testable statistical implications from causal DAGs (PEARL et al., 2000). Thus, we begin by mapping the existing knowledge to infer causality, even if they come from association studies.

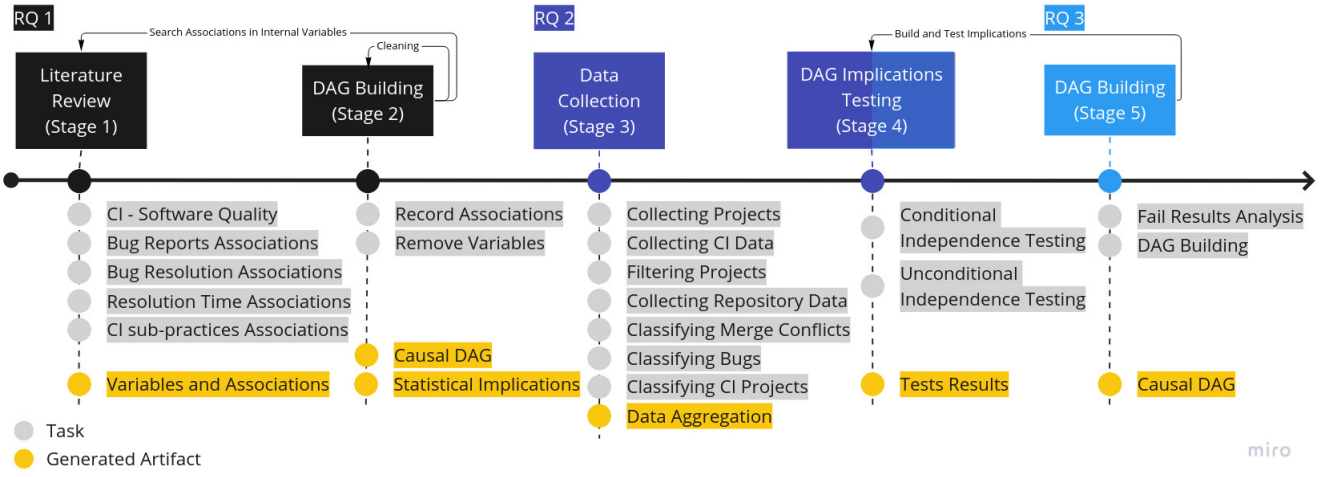


Figure 15 – Research method pipeline.

4.1.1 What does the literature proclaim about CI and software quality?

As depicted in Fig. 15, we perform two research stages to answer RQ1. First, the literature review compiles the existing assumptions about how CI may impact software quality and allow us to map variables, associations, and every common causes (i.e., the *Fire* in the example of Fig. 2(b)) for any two or more variables in the DAG. This mapped information is an input to stage 2 - DAG building. We perform these two stages iteratively to build a causal DAG representing the most up-to-date literature knowledge.

4.1.1.1 Literature Review

To review the relationship between continuous integration (CI) and software quality (SQ), we consider "issues", "bugs", and/or "defects" as proxies of software quality due to several compelling reasons. Firstly, bug reports serve as a valuable source of empirical data reflecting real-world user experience and interactions with a software system. By leveraging bug reports, we gain a practical means of assessing software quality that aligns with the user-centric perspective and provides valuable insights. Other studies, such as Vasilescu et al. (VASILESCU et al., 2015) and Santos et al. (SANTOS; COSTA; KULESZA, 2022), also used bug reports to indicate quality. Therefore, since we are also interested in knowing the common causes (i.e., the *Fire* in Fig. 2(a)) between CI and these variables (i.e., 'issues", "bugs", or "defects"), we review the literature for each one of these variables.

We also review the literature for each new variable found during the DAG building in an iterative manner. We perform the searches on Google Scholar[1] using the name of variables or synonyms. To cover a broader range of studies and mitigate search bias, we prioritize systematic literature reviews on the search. For instance, we search for software bug relationships using the search string "systematic literature review software bugs". This search has a broader perspective and not necessarily the studies mention CI, but it is essential to detect correlated variables and their relationships.

In addition, we search the literature for relationships among the following CI sub-practices — tests practices, integration frequency, build health maintenance, and quick fixes of broken builds (BECK; ANDRES, 2004; DUVALL; MATYAS; GLOVER, 2007; FOWLER, 2020). In the full search, we selected 39 studies, of which 12 are systematic literature reviews. All the selected studies are referenced, and a complete list is available in our replication package [2]. We believe our literature review fits our goal of finding the claims surrounding CI and SQ, especially because of the systematic literature reviews, as they have already systematically compiled a comprehensive view of the areas of CI and SQ.

[1] https://scholar.google.com/

[2] https://github.com/elieziosoares/ci_quality_study_replication

Table 20 – Connections identified in the literature about CI and software quality variables.

Connection	**Rationale**
$CI \rightarrow BugResolution$	Projects present more resolved issues and bugs after adoption of CI (RAHMAN et al., 2018).
$CI \rightarrow ResolutionTime$	CI is related to an increasing in the number of issues closed by period, helping to spend less time debugging and more time adding features (KAYNAK; ÇILDEN; AYDIN, 2019) (ZHAO et al., 2017).
$CI \rightarrow BugReport$	CI teams discover more bugs than no-CI teams, and CI projects present fewer defects than no-CI projects (AMRIT; MEIJBERG, 2017b; VASILESCU et al., 2015).
$CI \rightarrow Transparency$	CI is associated with a transparency increase, facilitating collaboration (KERZAZI; KHOMH; ADAMS, 2014).
$CI \rightarrow Communication$	The general discussion, the number of line-level review comments, and change-inducing review comments tend to decrease after CI adoption without affecting pull request activity (CASSEE; VASILESCU; SEREBRENIK, 2020).
$CI \rightarrow Overconfidence$	CI developers are reported as suffering from a false sense of confidence (when blindly trusting the tests) (PINTO; REBOUÇAS; CASTOR, 2017) (PINTO et al., 2018).
$CI \rightarrow TechnicalChallenges$	Configuring the build environment, the tools, and practices impose challenges for CI teams (PINTO; REBOUÇAS; CASTOR, 2017) (DEBBICHE; DIENéR; SVENSSON, 2014b).
$CI \rightarrow TestsVolume$	CI is associated with an increase in the test ratio (NERY; COSTA; KULESZA, 2019).
$CI \rightarrow CommitFrequency$	CI is linked to a change in the commits pattern (ZHAO et al., 2017) (RAHMAN et al., 2018).

Continuous Integration and Software Quality: Soares et al. (SOARES et al., 2022) conducted a systematic literature review on the associations between CI and software development as a whole. The review highlights associations between CI and an increase in bug/issue resolution (RAHMAN et al., 2018). For this reason, our literature-based DAG starts from the connection between continuous integration and bug resolution. The notation $CI \rightarrow BugResolution$ represents a causal flow from CI to Bug Resolution. We expand our DAG based on other studies that raise other diverse CI associations (as summarized in Table 20).

Confounders related to bug reports: To investigate potential confounders to the effect of $CI \rightarrow BugReport$, we search for other factors associated with Bug Rerport. Table 21 shows the associations extracted from a taxonomy by Huang et al. (HUANG; LIU; HUANG,), a systematic literature review by A. Cairo et al. (CAIRO; CARNEIRO; MONTEIRO, 2018), and a mixed mining software repositories (MSR)-survey study by Vasilescu et al. (VASILESCU et al., 2015).

Table 21 – Connections identified in the literature about bug reports.

Connection	**Rationale**
$LackOfKnowledge \rightarrow BugReport$	Insufficient domain and linguistic knowledge are presented as possible human root causes for software defects (HUANG; LIU; HUANG,).
$LackTechKnowledge \rightarrow BugReport$	Insufficient programming and strategy knowledge and failure to catch the specific feature of the problems are mapped as possible human root causes for software defects (HUANG; LIU; HUANG,).
$RequiremProblem \rightarrow BugReport$	Requirement management problems and a misunderstanding of requirements and design specifications are reported as possible human causes of software defects (HUANG; LIU; HUANG,).
$Overconfidence \rightarrow BugReport$	Overconfidence and confirmation bias contributes to evaluation errors and software defects (HUANG; LIU; HUANG,).
$Inattention \rightarrow BugReport$	Interruptions and other kinds of inattention are reported as possible human causes of software defects (HUANG; LIU; HUANG,).
$Communication \rightarrow BugReport$	Communication problems lead to expression and comprehension errors (HUANG; LIU; HUANG,).
$ConfigManagement \rightarrow BugReport$	Configuration management problems lead to process errors (HUANG; LIU; HUANG,).
$Tools \rightarrow BugReport$	Tools problems like compiler induced defects are possible root causes of software defects (HUANG; LIU; HUANG,).
$CodeSmells \rightarrow BugReport$	Code smells on the occcurrence of bugs (CAIRO; CARNEIRO; MONTEIRO, 2018).
$NumberOfForks \rightarrow BugReport$	The number of forks has an association with an increase in bug reports (VASILESCU et al., 2015).
$ProjAge \rightarrow BugReport$	Project age has a significant negative effect on the count of bugs reported by core developers (VASILESCU et al., 2015).
$ProjPopularity \rightarrow BugReport$	Project's popularity has a significant negative effect on the count of bugs reported by core developers (VASILESCU et al., 2015).
$QuantIssues \rightarrow BugReport$	The number of non-bug issue reports has a significant and positive effect on the response (VASILESCU et al., 2015).
$TestsVolume \rightarrow BugReport$	The size of test files has a negative effect on bug reports (VASILESCU et al., 2015).

Confounders related to bug resolution: To investigate potential confounders to the effect of $CI \rightarrow BugResolution$, we search for other factors associated with $BugResolution$ in the literature. We found that $Maintainability$, $Analysability$, $Changeability$, $Stability$, $Testability$, $ProjectVolume$, $Duplication$, $UnitSize$, $UnitComplexity$, and $ModuleCoupling$ all share an association with $BugResolution$. For the sake of readability, we group all these relationships into $InternalQuality \rightarrow BugResolution$ (FERREIRA et al., 2012). We also found the associations $Communication \rightarrow BugResolution$ and $IssuePriority \rightarrow BugResolution$ in the literature review from Zhang et al. (ZHANG et al., 2016). Table 22 shows all these associations and their rationales.

Table 22 – Connections identified in the literature about bug resolution.

Connection	**Rationale**
$InternalQuality \rightarrow BugResolution$	Elements of $InternalQuality$, such as maintainability, analysability, changeability, stability, testability, project volume, duplication, unit size, unit complexity, and module coupling, present significant correlation with defect resolution efficiency (HUANG; LIU; HUANG,).
$Communication \rightarrow BugResolution$	Human and data elements such as comments, severity, product, component, among others, can improve the performance of bug resolution (ZHANG et al., 2016).
$IssuePriority \rightarrow BugResolution$	Priority and severity are non-textual factors of a bug report that enhance the capability of bug resolution (ZHANG et al., 2016).

Confounders related to resolution time: Table 23 shows associations between other factors than CI and $ResolutionTime$. We found associations extracted from the literature including variables $IssueType$, $Communication$ (e.g., comments in issues, bug reports, pull requests), $IssuePriority$, $CommitFrequency$, $OperateSystem$, and $IssueDescription$.

Table 23 – Connections identified in the literature about the resolution time.

Connection	**Rationale**
$IssueType \rightarrow ResolutionTime$	Issue fixing times are different for different issue types (MURGIA et al., 2014; LICORISH; MACDONELL, 2017) (ZHANG et al., 2012; MOCKUS; VOTTA, 2000).
$Communication \rightarrow ResolutionTime$	The number of comments and the max length of all comments in the bug reports impact the resolution time. Bugs with little discussion tend to be resolved quickly (PANJER, 2007) (ZHANG et al., 2012).
$IssuePriority \rightarrow ResolutionTime$	The severity of a bug report influences the delay before fixing it. As high the severity level, the fewer the delay (ZHANG et al., 2012).
$CommitSize \rightarrow ResolutionTime$	The size of code churn (number of methods) impacts the delay before fixing a bug report (ZHANG et al., 2012).
$OperateSystem \rightarrow ResolutionTime$	The median delay before fixing a bug found on Linux is shorter than other OS (ZHANG et al., 2012).
$IssueDescription \rightarrow ResolutionTime$	Increasing the literal length of the bug report description can increase delay until the team checks it as resolved (ZHANG et al., 2012).

Internal confounders: We also consider internal relationships between every discovered variable to understand possible confounding scenarios in our causal DAG. That means considering potential associations between peripheral variables, e.g., $IssueType \rightarrow CommitFrequency$. Table 24 shows such potential associations and their rationales.

Table 24 – Internal associations cataloged among the literature regarding the discovered variables.

Association	**Rationale**
$IssueType \rightarrow CommitSize$	The issue type is associated with the size of the code churn (HINDLE; GERMAN; HOLT, 2008).
$IssueType \rightarrow Engagement$	Developers tend to spend more effort engaging with one another regarding new features and software extensions than in defects (LICORISH; MACDONELL, 2017).
$IssueType \rightarrow InfoSharing$	Developers tend to share more information on defects and enhancements than support tasks (LICORISH; MACDONELL, 2017).
$IssueType \rightarrow Communication$	A higher number of comments is associate with enhancements and defects (LICORISH; MACDONELL, 2017).
$IssueType \rightarrow DifficultyLevel$	There is an association between the difficulty of a change and its type (MOCKUS; VOTTA, 2000).
$Stability \rightarrow TechnicalChallenges$	The maturity of the tools, infrastructure, and CI activities imposes challenges to practitioners (DEBBICHE; DIENéR; SVENSSON, 2014b). The stability and maturity of the software under test affect the maintenance effort of tests (GAROUSI; MäNTYLä, 2016).

With these previous connections, we build the partial causal DAG represented in Fig. 16, where continuous integration is the intervention, and *BugResolution* and *BugReport* are potential outcomes.

Confounders related to Continuous Integration and its sub-practices: To investigate variables related to continuous integration (CI), we also consider its sub-practices (BECK; ANDRES, 2004; DUVALL; MATYAS; GLOVER, 2007; FOWLER; FOEMMEL, 2006). We consider the practices: automated tests practices, integration frequency, build health maintenance, and time to fix a broken build (DUVALL; MATYAS; GLOVER, 2007). We represent an association with a sub-practice of CI as an association with CI itself. For example, $AutomatedTests \rightarrow Confidence$ will be represented as $ContinuousIntegration \rightarrow Confidence$. This decision avoids intangible discussions about what comes first, CI or Automated Tests, for instance. To implement CI, we consider that such practices are implicit, i.e., there is no CI practice without implementing those sub-practices (SOARES et al., 2022; DUVALL; MATYAS; GLOVER, 2007). Therefore, distinguishing the effect of CI and the effect of its sub-practices would be challenging. Table 25 shows the associations relating to testing, while Table 26 groups and presents those relating build practices.

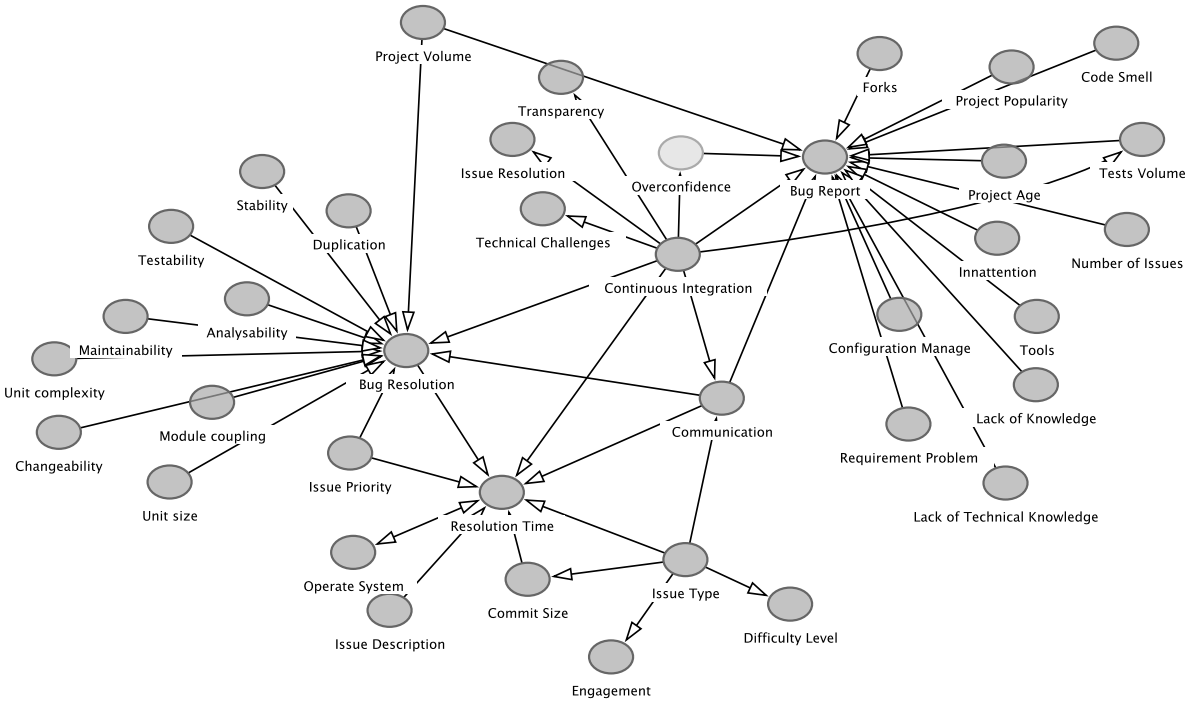


Figure 16 – Partial causal DAG for bug reports associations.

Table 25 – CI associations cataloged among the literature from the perspective of test practices.

Association	**Rationale**
$AutomatedTests \rightarrow BugReport$	Automated tests are related to improved product quality in terms of fewer defects in the software (RAFI et al., 2012).
$AutomatedTests \rightarrow CodeCoverage$	Automated tests are related to high coverage of code (RAFI et al., 2012).
$AutomatedTests \rightarrow WorkTime$	Automated tests are related to reduced testing time (RAFI et al., 2012).
$AutomatedTests \rightarrow Confidence$	Automated tests are related to increased confidence in the quality of the system (RAFI et al., 2012).
$AutomatedTests \rightarrow HumanEffort$	Automated tests are related to the less human effort that can be redirected for other activities (RAFI et al., 2012).
$AutomatedTests \rightarrow Cost$	Automated tests are related to a reduction in cost (RAFI et al., 2012).
$AutomatedTests \rightarrow BugDetection$	Automated tests are related to increased fault detection (RAFI et al., 2012).
$AutomatedTests \rightarrow TechnicalChallenges$	Automated tests require different skills to implement them effectively (GAROUSI; MäNTYLä, 2016).
$LackTechKnowledge \rightarrow AutomatedTests$	The skills level of testers could be a hindrance to test automation (GAROUSI; MäNTYLä, 2016).
$TestDesign \rightarrow TestReusability$	Designing tests with maintenance in mind, they can be repeated frequently (RAFI et al., 2012).
$TestRepetition \rightarrow Reliability$	When repeating tests, they are more reliable than single executions (RAFI et al., 2012).
$ProjAge \rightarrow AutomatedTests$	The number, coverage, and maturity of automated tests increase with time (ZAIDMAN et al., 2008; ZAIDMAN et al., 2011) (HILTON; BELL; MARINOV, 2018; NERY; COSTA; KULESZA, 2019).

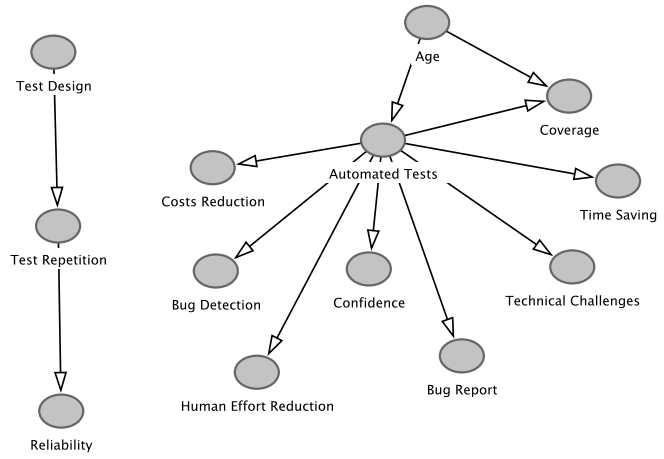


Figure 17 – Partial causal DAG for automated tests associations.

With connections shown in Table 25 we build the partial causal DAG represented in Fig. 17, where $AutomatedTests$ is the intervention. The relationship $AutomatedTests \rightarrow BugReport$ was omitted in this figure because it is a partial DAG centered in the $AutomatedTests$. With Table 26, we build the partial causal DAG represented in Fig. 18, centered in the build attributes.

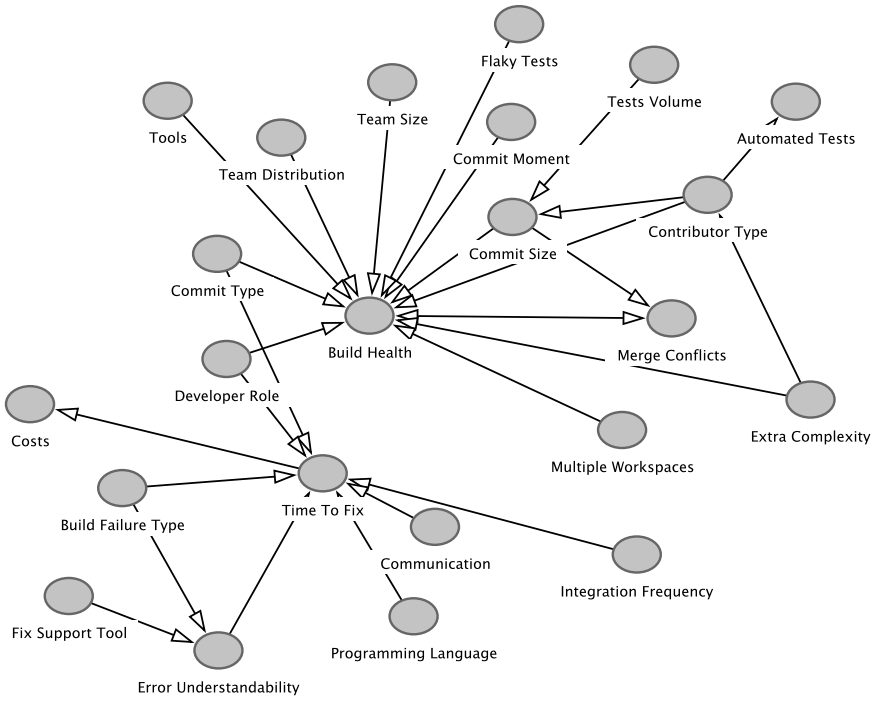


Figure 18 – Partial causal DAG for build attributes and their associations.

Fig. 19 shows the unified and complete literature-based causal DAG, i.e., the union of the assumptions cataloged in Tables 20, 21, 22, 23, 24, 25, 26. The literature-based causal DAG expresses all known associations for the related variables in the studied domain. The nodes on the DAG represent variables (e.g., Continuous Integration and Bug Report), and the directed edges connecting the variables represent an association between them.

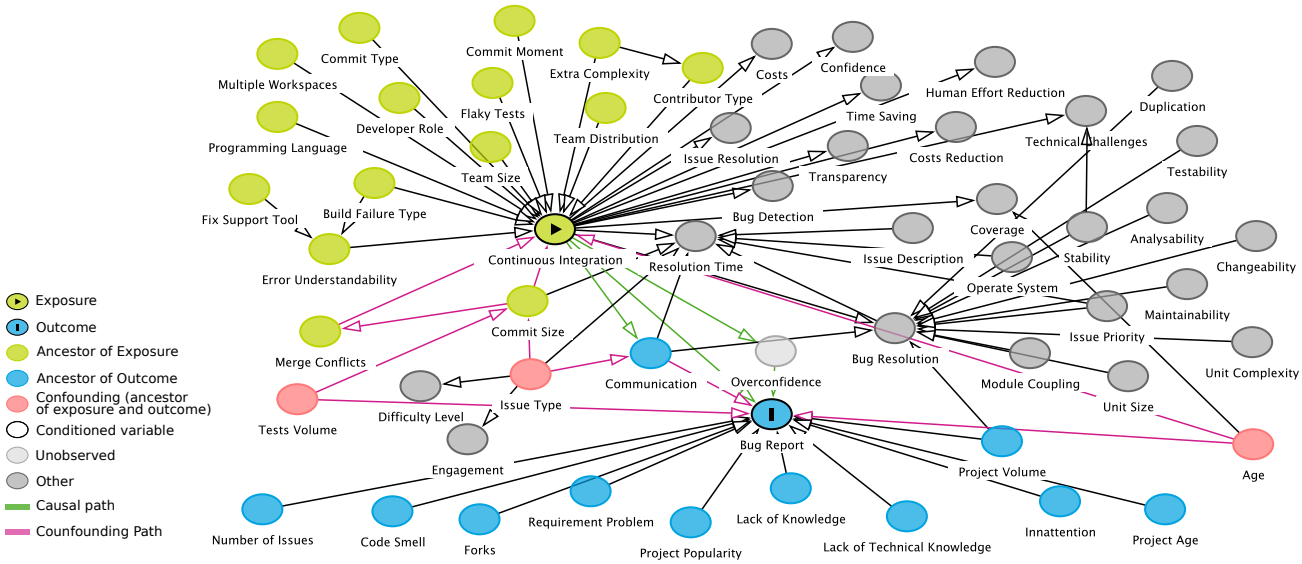


Figure 19 – Complete literature-based causal DAG for CI, Bug Reports and their co-variables.

The associations "flow" from one variable to another. For instance, according to Fig. 19, Continuous Integration influences Bug Report, i.e., CI teams discover more bugs and CI projects present fewer defects than NOCI projects (those that do not adopt CI) (AMRIT; MEIJBERG, 2017b; VASILESCU et al., 2015). Such DAG associations could represent a positive or a negative association, and whenever possible, this interpretation is discussed in the text. Note that the variables $AutomatedTests$, $BuildHealth$, $IntegrationFrequency$, and $TimeToFix$ are all represented by $ContinuousIntegration$. This causal DAG built upon the existing literature is the starting point for identifying variables that must be measured and controlled to allow a causal analysis of the influence of CI on software quality.

Table 26 – CI associations cataloged among the literature from the perspective of build practices.

Association	**Rationale**
$BuildHealth \rightarrow WorkTime$	Broken builds lead to loss of time by freezing development and tests (KERZAZI; KHOMH; ADAMS, 2014).
$BuildHealth \rightarrow MergeConflicts$	Broken builds lead to work blockage, which in turn leads to merge conflicts (LAUKKANEN; ITKONEN; LASSENIUS, 2017).
$TeamSize \rightarrow BuildHealth$	The team size relates to build breakage. Shorter teams tend to break fewer than larger ones (KERZAZI; KHOMH; ADAMS, 2014).
$MultipleWorkspace \rightarrow BuildHealth$	Maintaining multiple physical structures for multiple branches is associated with more build breakage (KERZAZI; KHOMH; ADAMS, 2014).
$DeveloperRole \rightarrow BuildHealth$	There is a statistical difference in build breakage among different role groups (KERZAZI; KHOMH; ADAMS, 2014).
$CommitSize \rightarrow BuildHealth$	The size of the changes is related to a higher probability of build failure (RAUSCH et al., 2017; ISLAM; ZIBRAN, 2017) (KERZAZI; KHOMH; ADAMS, 2014).
$CommitType \rightarrow BuildHealth$	The commit type (such as features and bugs) and the contribution model (e,g., pull request and push model) are associated with build breakage (KERZAZI; KHOMH; ADAMS, 2014; RAUSCH et al., 2017) (ISLAM; ZIBRAN, 2017).
$CommitMoment \rightarrow BuildHealth$	There is an association between the moment of contributions and the rate of build breakage (KERZAZI; KHOMH; ADAMS, 2014).
$TeamDistribution \rightarrow BuildHealth$	The geographical distance of the team members is associated with the build results (KERZAZI; KHOMH; ADAMS, 2014).
$Tools \rightarrow BuildHealth$	The languages and their tools are related to different build breakage rates (SEO et al., 2014).
$ExtraComplexity \rightarrow BuildHealth$	Complex builds tend to break (LAUKKANEN; ITKONEN; LASSENIUS, 2017).
$FlakyTests \rightarrow BuildHealth$	Flaky tests favor the occurrence of build breakage (LAUKKANEN; ITKONEN; LASSENIUS, 2017)(RAUSCH et al., 2017).
$ContributorType \rightarrow BuildHealth$	Less frequent contributors tend to break builds less (RAUSCH et al., 2017).
$TimeToFix \rightarrow Costs$	The time lost relates directly to a monetary cost (KERZAZI; KHOMH; ADAMS, 2014).
$Communication \rightarrow TimeToFix$	The feedback mechanisms and information speed affect the awareness of a broken build and the time to fix it (KERZAZI; KHOMH; ADAMS, 2014).
$DeveloperRole \rightarrow TimeToFix$	The developer role is associated with the time to fix a broken build (KERZAZI; KHOMH; ADAMS, 2014).
$CommitType \rightarrow TimeToFix$	The characteristics of the branches and code access (e.g., isolated branches) are associated with the time to fix a broken build (KERZAZI; KHOMH; ADAMS, 2014).
$IntegrationFreq \rightarrow TimeToFix$	The integration frequency in the team affects the build fixing (KERZAZI; KHOMH; ADAMS, 2014).
$ProgramLanguage \rightarrow TimeToFix$	The programming language is related to the time spent to fix a broken build (SEO et al., 2014).
$ErrorUnderstand \rightarrow TimeToFix$	The understandability of the build failures directly impacts the time needed to solve them (VASSALLO et al., 2020).
$BuildFailType \rightarrow TimeToFix$	The build failure types are associated with different difficulty levels (VASSALLO et al., 2020).
$TestsVolume \rightarrow CommitSize$	Complex and time-consuming testing is a possible reason for large commits (LAUKKANEN; ITKONEN; LASSENIUS, 2017).
$ContributorType \rightarrow CommitSize$	The type of contributor (e.g., casual) relates to the build breakage rate (REBOUCAS et al., 2017).
$ContributorType \rightarrow AutomatedTests$	The contributor type is related to the number of automated tests (REBOUCAS et al., 2017).
$Extracomplexity \rightarrow ContributorType$	The complexity of the jobs is related to the type of contributor in the projects (REBOUCAS et al., 2017).
$CommitSize \rightarrow MergeConflicts$	Large commits are associated with merge conflicts (LAUKKANEN; ITKONEN; LASSENIUS, 2017).
$FixTools \rightarrow ErrorUnderstand$	Fix support tools improves the understandability of the build logs (VASSALLO et al., 2020).
$BuildFailType \rightarrow ErrorUnderstand$	The build failure type is associated with different levels of understandability (VASSALLO et al., 2020).

4.1.1.2 DAG Building

To analyze the causal effect of CI on software quality, we need to be attentive to all potential confounding effects, i.e., bias due to backdoor paths (see Section 2.4.2). Thus, we draw a DAG containing a sufficient set of variables that show the existing backdoor paths with respect to CI and software quality. This new DAG is a subgraph of the DAG shown in Fig. 19.

Based on Reichenbach's Common Cause Principle and the Markov Condition, we know that a causal DAG should include the common causes of any pair of variables in the DAG (PEARL et al., 2000; HERNáN; ROBINS, 2010). Therefore, we can discard several variables in Fig. 19 because they are external variables without connecting with the variables under investigation (i.e., CI and $BugReport$), and they are not common causes for any variable selected for the analysis (i.e., CI, Bug Report or one of its common causes). For instance, $ProjectPopularity$ is associated with $BugReport$ but does not have an association with no other variable on the DAG. Thus, $ProjectPopularity$ can be discarded from our causal analysis. On the other hand, Age is a common cause associated with $ContinuousIntegration$ and $BugReport$, and it is essential to analyze the causal effect between them since Age is a potential confounding factor (see Section 2.4.2).

4.1.2 RQ2. Is the causal effect of CI on software quality empirically observable?

Based on the concepts of conditional independence and the d-separation rules (see Section 2.4.1), we analyze a set of statistically testable restrictions as implications of a model (PEARL et al., 2000). Such restrictions (i.e., statistical implications) are conditional or unconditional independencies between DAG variables that must be found in any dataset generated by the causal processes described in the DAG. By building a dataset with the variables in our causal DAG, we can test the implications of the DAG statistically. Note that because these statistical tests are based on the d-Separation rules, we are not only checking for associations but also for causal relationships (i.e., considering the "flow" between relationships in the DAG) (PEARL; VERMA, 1995; PEARL; JUDEA, 1994; PEARL et al., 2000). We are discovering the causal structure and inferring causation from the initial DAG built in the RQ1 (section 4.1.1) and the testing of their d-Separation implications on an empirical data set (SHALIZI, 2021).

The implications are in the form of unconditional independencies, like $Age \perp\!\!\!\perp TestsVolume$, which means Age is independent of $TestsVolume$. Alternatively, the implications may have the form of conditional independencies, like $MergeConflicts \perp\!\!\!\perp TestsVolume \mid CommitFrequency$, which means $MergeConflicts$ is independent of $TestsVolume$ conditioned in $CommitFrequency$. Since we use the `DAGitty R` package (TEXTOR et al., 2016) to draw our causal DAGs, we obtain from the

`impliedConditionalIndependence` function a list of testable implications that become our causal hypotheses.

We then collect data by mining software repositories to empirically analyze the selected variables, allowing us to test the causal DAG and its statistical implications, i.e., our causal hypotheses.

4.1.2.1 Collecting Data & Empirical Analysis

Based on the knowledge and assumptions acquired in the literature review and the causal DAG built to answer the RQ1, we have support for building a dataset to analyze the relationship between CI and software quality, including confounding variables. Using the dataset, we can check the validity of our DAG through statistical tests. Fig. 20 summarizes the dataset building process, and we detail such process in the following. All tools, scripts, and information necessary to reproduce the process described in the sequence are available in our replication package [3].

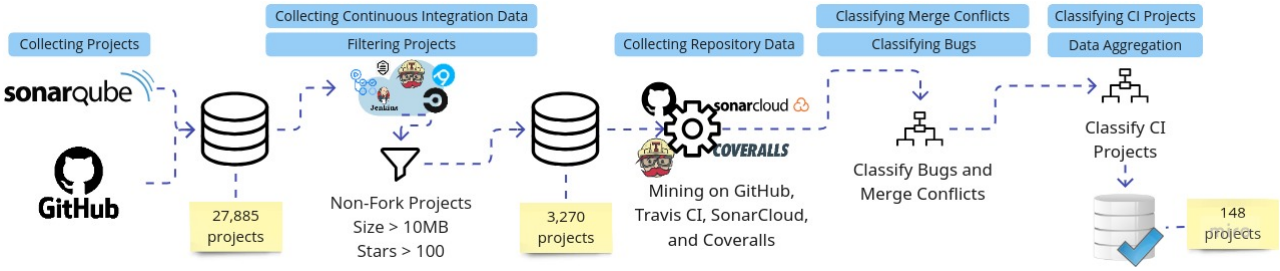


Figure 20 – Mining Software Repository Process

Collecting Projects. As illustrated in Fig. 20, we start with a list of 27,885 projects obtained from the most starred repositories in the GitHub Search API[4] and a list of projects' names from the SonarQube web API[5]. We searched for projects having a public repository on GitHub to start the mining repository process by collecting the history of developers' contributions to them. The search included 15 popular languages: C, C#, C++, Go, Java, JavaScript, Kotlin, Objective-C, PHP, Python, Ruby, Rust, Scala, Swift, and TypeScript. So the only initial premise was to have a public repository on GitHub.

Collecting Continuous Integration Data. In the next steps, we search for data in continuous integration services. Inspired by Hilton's work (HILTON et al., 2016), which searches for the usage of 5 different services — TravisCI, CircleCI, AppVeyor, Werker, Cloud-Bees, and Jenkins CI, we search for the same services and add a new one — GitHub

[3] https://github.com/elieziosoares/ci_quality_study_replication
[4] https://docs.github.com/en/rest/reference/
[5] https://community.sonarsource.com/t/list-of-all-public-projects-on-sonarcloud-using-api/33551

Actions[6]. We discovered a high CI service usage (54.3%) among the collected projects. Travis CI [7] appeared as the most popular service in our dataset.

Despite the evidence regarding a decreasing usage of Travis CI service (WIDDER et al., 2018; DECAN et al., 2022; GOLZADEH; DECAN; MENS, 2022), it is worth mentioning that Travis is still the most popular service in our dataset (see Table 27), which has also been the case for recent research analyzing machine learning projects using CI (RZIG et al., 2022). As for GitHub Actions, although the relatively short time since its launch (November 2019), it has been used by a significant number of projects up to our study's data collection date (2022). This observation confirms the findings from Decan et al. (DECAN et al., 2022). When they analyzed 67,870 GitHub repositories, they found that more than 4 out of 10 use GitHub Actions workflows. Golzadeh et al. (GOLZADEH; DECAN; MENS, 2022) also observed that while the adoption rate of the other CI services decreased, GitHub Actions has a steadily increasing adoption rate.

Table 27 displays the identified Continuous Integration (CI) services, the number of projects categorized as users for each service, and the criteria employed to categorize and classify the adoption of these services.

We build scripts to identify which CI service is used by each project. For Travis CI, Circle CI, and Wercker, we check the public APIs of these services by searching for the project name and organization. Specifically for Travis CI, if we do not find any information through the Travis API, our scripts search for the *".travis.yml"* file using the GitHub API. We also keep track of when the service configuration was initiated, specifically noting the date of the first commit of the *".travis.yml"* file.

Table 27 – The CI service usage on the dataset and the classification criteria.

CI Service	**Qty**	**Criteria**
Travis CI	9,092 (32.6%)	We search on Travis API[a] for the existence of the project in the service. If true, we search on GitHub for the existence of a *".travis.yml"* file.
GitHub Actions	5,630 (20.1%)	Using the search code feature of GitHub API, we search for the extension *".yml"* in the path *".github/workflows"*.
Circle CI	307 (1.1%)	Through Circle CI API[b] we search for the existence of the project on the service. If not located, we searched for the file *"config.yml"* in the path *".circleci"*.
Jenkins	58 (0.2%)	Using the search code feature of GitHub API, we search for the file *"Jenkinsfile"*.
AppVeyor	29 (0.1%)	Using the search code feature of GitHub API, we search for the file *"appveyor.yml"*.
Wercker	0	Using the Wercker API[c] we search for the existence of the project on the service.
No CI Service	12,769 (45.7%)	

[a]https://docs.travis-ci.com/user/developer/#api-v3
[b]https://circleci.com/docs/api/v2/
[c]https://devcenter.wercker.com/development/api/endpoints/

[6] https://github.com/features/actions
[7] https://travis-ci.org/

For the remaining services, the script checks if the configuration file exists. In the case of AppVeyor, we verify the presence of a file named *"appveyor.yml"* within the GitHub repository using the API. For Jenkins, we search for a *"Jenkinsfile"*, while for Circle CI, we look for a *"config.yml"* inside a *".circleci"* folder. Similarly, for GitHub Actions, we examine whether a file with the extension *".yml"* exists inside a *".github/workflows"* folder.

Filtering projects. Afterwards, we filtered out irrelevant repositories as represented in Fig. 20. In particular, we consider only non-fork projects with more than 100 stars. Despite the risks of applying MSR in empirical studies and selecting non-representative projects (MUNAIAH et al., 2017), we selected "engineered projects" as opposed to toy projects (MUNAIAH et al., 2017) and applied a series of criteria, beginning by excluding projects smaller than 10MB (OLIVEIRA, 2017; OLIVEIRA et al., 2019). These two criteria are the initial steps towards selecting projects that approach the "engineered projects" concept (MUNAIAH et al., 2017). While mining data from the repository and classifying projects for analysis, we ensure a rigorous selection of projects that represent engineered projects. In the sequence, we detail each step of mining and classification of projects.

Next, we consider only projects using either Travis CI or projects not using a CI service. Including projects not using a CI service is essential because we create a control group of no-CI projects in order to understand the effects on the CI projects group. As mentioned earlier, we selected Travis CI as a target because it is the most used service in our dataset. In addition, Travis CI has a public API to obtain software builds data. After this filtering step, our sample was reduced from 27,885 to 3,270 repositories, i.e., 2,527 repositories using Travis CI and 743 not using a CI service.

Collecting Repository Data. We mine the pull requests and issues of the projects, filtering out those projects that do not have pull requests or issues. We collect data from pull requests, issues, commits, and comments using the GitHub Search API. We use the Travis API to collect data from software builds, while we use two different services (Coveralls [8] and SonarCloud [9]) to search for code coverage information.

We collected pull requests and issues in the repositories using the endpoint "pulls" and "issues" from the GitHub Search API. This process resulted in a total of 1,425,493 pull requests and 3,328,221 issues. Next, we collected all pull request comments and associated information regarding authors. All these data allow us to compute a metric to measure *Communication* detailed forward.

We also collect each pull request's commit, sha, date, message, and author. To collect the detailed commit information regarding lines of code added, removed, or changed, we request it to GitHub API [10]. We processed the data to obtain the size of a commit, the number of lines of added/removed code, the number of files modified, and the test files and test lines. The commits data help us to compute metrics to measure $CommitFrequency$,

[8] https://coveralls.io/
[9] https://sonarcloud.io/
[10] https://api.github.com/repos/{owner}/{repo}/pulls/{pull_number}/files

$TestsVolume$, and $MergeConflicts$.

Classifying Merge Conflicts. To identify the merge conflicts occurrences, we used an algorithm (see Algorithm 1) to reconstruct the commits history and identify a merge conflict when it was generated. The algorithm clones the repository (line 1) and gets the list of commits (line 4). The algorithm gets the parents for each commit and verifies the number of parent commits (line 6). If a commit has more than one parent, we call the `git diff` function to them (line 7). Then, depending on the `diff` result, we can detect a conflict (line 9) and append it to the list of merge conflicts (line 10). We manually validated the algorithm with a random sample of 40 commits and achieved an accuracy of 90%.

Algorithm 1: Detect Merge Conflicts

Require: *remote*: URL for the project repository
Require: *repository_path*: Local path to clone the project repository

```
1:  repo ← Repo.clone_from(remote, repository_path)
2:  merge_conflicts ← ∅
3:
4:  for all commit in repo.commits() do
5:
6:    if commit.parents > 1 then
7:      merge_diff ← repo.git.diff(commit.parents[0], commit.parents[1])
8:
9:      if merge_diff contains " <<<<<<< HEAD" and " >>>>>>> " then
10:       Append commit.sha to merge_conflicts
11:
12:     end if
13:
14:   end if
15:
16: end for
17:
18: return merge_conflicts
19: End function
```

Classifying Bugs. To classify issues as bugs, we consider only projects using GitHub issues and labeling them. We mapped the labels used in their repositories through a semi-manual inspection, i.e., we began with a script to a preliminary parse relying on a list of bug-related keywords (VASILESCU et al., 2015; SANTOS; COSTA; KULESZA, 2022). Then, we manually validated the labels for each repository using GitHub issues that have assigned to them. This approach is similar to the approach used by Vasilescu et al. (VASILESCU et al., 2015) and Santos (SANTOS; COSTA; KULESZA, 2022).

The exception to our primary approach is a set of 10 projects with many issues without labels. To avoid missing these data, we adopt a conservative approach classifying as bugs issues containing the terms “bug” or “fix” in their title or their body.

Classifying CI Projects. In order to avoid the CI Theater (FELIDRé et al., 2019; THOUGHTWORKS, 2017) and to adopt recommendations from Soares et al. (SOARES et al., 2022), in addition to the use of Travis CI, we consider build and code coverage information, meaning that we select projects that have actual build and test activity. After achieving a set of 74 CI projects surviving all filtering stages, we randomly drew 74 out of 95 no-CI projects to balance our dataset. Table 28 shows a summary of the final dataset.

Data Aggregation. With the collected data, we can analyze dimensions of releases such as Commit Frequency, Test Volume, Merge Conflicts, Communication, Bug Reports, and Age variables. We cannot measure Issue Type and Overconfidence for two reasons. First, classifying $IssueType$ from the collected issue data is challenging given that projects do not have consistent patterns for classifying issue patterns (e.g., standardized tags, such as "enhancement", "perfective", "corrective" used in a standardized manner). Second, overconfidence is a subjective feeling not feasible to measure through collecting data from repositories. Therefore, we consider these two variables as *latent* (i.e., unmeasured). Latent variables can still be represented and analyzed in a causal DAG, even if they cannot be statistically tested. However, we can still interpret these latent variables based on the statistical tests of the other variables within the DAG (HERNáN; ROBINS, 2010).

In the DAG and the dataset, we used the variable Commit Frequency, replacing Commit Size. These variables have an intrinsic relationship since more frequent commits tend to be smaller. However, it is noteworthy that commit frequency is most noticeable in the context of projects using Travis CI (ZHAO et al., 2017). In addition, more frequent commits engender a cascade of CI builds and verifications, potentially detecting bugs earlier, as cited in (SOARES et al., 2022). In light of this, we employed a commit frequency metric to build a more consistent and complete DAG.

We derive the metrics by aggregating their values by project releases. We collected data covering 12 development months, similar to data analyzed by Zhao et al. (ZHAO et al., 2017) and Santos et al.(SANTOS; COSTA; KULESZA, 2022). Notably, all 148 projects under scrutiny encompass a data-mined span of 12 months. We consider the first 12 months for CI projects starting from the month when CI was adopted. For no-CI projects, we consider the starting month for analysis according to the median *Age* of the CI projects, similar to Sizilio et al. (NERY; COSTA; KULESZA, 2019).

In addition to *Age*, the *Size* of the projects are also balanced. We tested this difference through a `Wilcoxon Rank` test and found no statistically significant difference between the two groups ($p - value = 0.7372$). Regarding the difference in bug reports, we also obtained a negligible difference ($p - value = 0.1318$). On the other hand, we found a difference between the two groups for the number of stars on GitHub ($p-value = 0.000679$), for the number of issues ($p - value = 0.007304$), and for pull-requests ($p - value = 1.737e{-}07$). In simple terms, the CI projects are more active and popular, but not necessarily with greater maturity and size.

For each project release in such time, we compute the following metrics:

- **Commit Frequency** represents the aggregate count of commits across all pull requests within a given release.
- **Communication** is the sum of comments and review comments in the pull requests. We consider the average of communication in a release.
- **Merge Conflicts** is the number of merge conflicts in a release. We traverse the repositories commit tree and use the GIT command diff to verify the merge commits and the conflicts.
- **Age** is the number of days of a repository from its creation until the release date.
- **Test Volume** is the proportion of lines of test code modified (added, removed, or changed) in the commits. We applied the strategy from Nery et al. (NERY; COSTA; KULESZA, 2019) to identify test files. We consider the median test volume in a release.
- **Bug Reports** is the number of bugs reported in a release.
- **CI** is a binary categorical variable indicating whether the project uses CI.

Table 28 – Summary of the Data Set.

Number of Projects	148
Number of Pull Requests	18,961
Number of Commits	59,034
Number of Builds	16,619
Number of Issues	41,866
Number of Bugs	3,199
Number of Merge Conflicts	403

4.1.2.2 DAG Implications Testing

Having the set of testable causal hypotheses obtained in RQ1 and the produced dataset (Section 4.1.2.1), we can investigate if the implications from the causal DAG are actually held in the empirical dataset. We test the unconditional independence causal hypotheses (e.g., $X \perp\!\!\!\perp Y$) using the `dcov.test` function from the `energy R` package [11]. This test verifies if two variables on the dataset are independent. We test the conditional independence causal hypotheses (e.g., $X \perp\!\!\!\perp Y \mid Z$) with the *Kernel conditional independence test* from the `CondIndTests R` package (HEINZE-DEML; PETERS; MEINSHAUSEN, 2018). Both tests are non-parametric and suitable for our data.

[11] https://cran.r-project.org/package=energy

4.1.3 RQ3.What would be an accurate causal theory for CI?

Considering the investigations in RQ1 and RQ2, we have information regarding which independence tests have passed or failed in our analyses. Such information allows us to propose adaptations to the causal DAG built in RQ1. For example, by testing the data, we can observe a dependency between *Age* and *Communication*, so we propose a new edge $Age \rightarrow Communication$. We build a new causal DAG by combining the knowledge from the literature (RQ1) and the statistical tests on the dataset (RQ2). We call this hybrid approach data-validated causal DAG. The data-validated causal DAG is compatible with the dataset in terms of the independency relationships between the variables. Then, we perform a data-validated approach for causal discovery.

There are dozens of algorithms for causal discovery from data, such as PC (SPIRTES; GLYMOUR; SCHEINES, 1993), that rely on conditional independence tests to discover the causal structure from the data. On the other hand, Cartwright (CARTWRIGHT, 1989) argues that causal investigation requires background information too (i.e., causal assumptions)—"No causes in, no causes out".

Thus, to implement the data-validated causal DAG, we start getting background information from the literature review and building the initial causal DAG in RQ1 (i.e., causal assumptions from the literature). Next, we combine the knowledge acquired in RQ2 (i.e., the independence relationships that were not present in the data), and finally, we structure our RQ3 procedure in three steps.

Step 1. By analyzing the rejected hypotheses in RQ2 (Section 4.2.2), in conjunction with d-separation rules (Section 2.4.1), we can infer connections that should be added or removed in the data-validated causal DAG.

Step 2. We generate the data-validated causal DAG after identifying the disconnected vertices and the necessary changes from Step 1.

Step 3. We test the d-separation implications on the data-validated causal DAG. If any test fails, we return to Step 1 and refine the DAG.

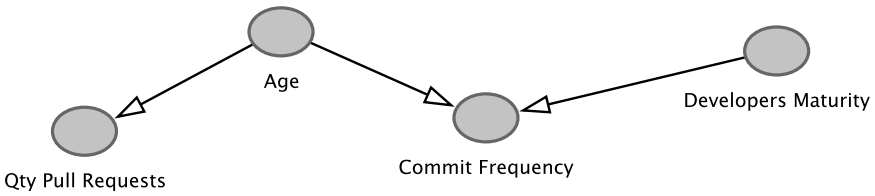


Figure 21 – Hypothetical causal DAG.

Fig. 21 presents a hypothetical DAG to illustrate the steps to obtain the data-validated DAG. The DAG contains a Fork structure (i.e., $QtyPullRequests \leftarrow Age \rightarrow CommitFrequency$) and a collider (i.e., $Age \rightarrow CommitFrequency \leftarrow DevelopersMaturity$.

Analyzing Forks and Chains. The d-Separation rules impose the same principle for the analysis of forks or chains (see Section 2.4.1), i.e., if this structure is true,

QtyPullRequests and *CommitFrequency* should be statistically dependent because an association flows through forks and chains. Otherwise, these variables should be independent when we block the path by conditioning on the middle variable *Age*, as in Fig. 22(a).

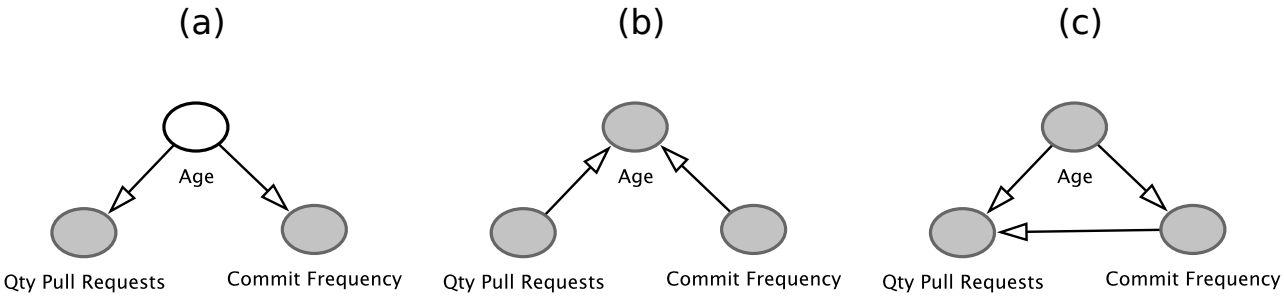


Figure 22 – Examples of causal DAG structure testing.

Therefore, we can statistically test the fork in Fig. 22. We should perform a conditional independence test to check if *QtyPullRequests* and *CommitFrequency* are independent conditioning on *Age* ($QtyPullRequests \perp\!\!\!\perp CommitFrequency \mid Age$) as in Fig. 22(a). If the test rejects such a hypothesis, meaning that the structure is wrong, we may hypothesize different structures.

For instance, "what if the structure would be a collider $QtyPullRequests \rightarrow Age \leftarrow CommitFrequency$? (see Fig. 22(b)). In this case, *QtyPullRequests* and *CommitFrequency* would be independent without conditioning. On the other hand, if the variables have a dependence that does not interrupt when conditioning on *Age*, they share a relationship passing through another path, then we add a new edge between $QtyPullRequests \leftarrow CommitFrequency$, as illustrated in Fig. 22(c).

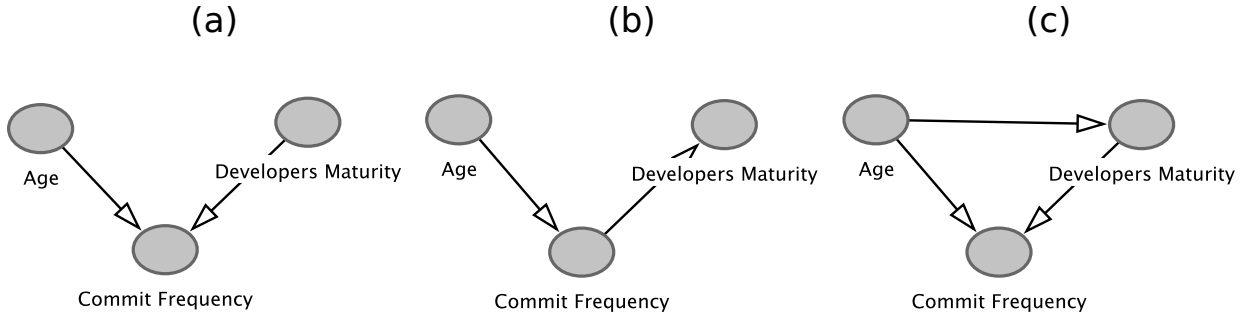


Figure 23 – Examples of causal DAG collider structure testing.

Analyzing Colliders. The d-Separation rules (see Section 2.4.1) imply that the variables in a collider are independent because the collider blocks the association flow, i.e., *Age* and *DevelopersMaturity* should be statistically independent. On the other hand, if we condition on *CommitFrequency*, they would be dependent because the condition opens the association flow in a collider.

We can statistically test the structure in Fig. 23(a) by testing the independence between *Age* and *DevelopersMaturity* ($Age \perp\!\!\!\perp DevelopersMaturity$). In case the variables

are not independent, it means that the structure is not a collider or there exists another path linking them. Thus, we can verify if the structure is a chain as illustrated in Fig. 23(b) by verifying if conditioning on $CommitFrequency$ they become independent. Finally, if the last hypothesis fails, i.e., Age and $DevelopersMaturity$ are not independent, even if conditioning on $CommitFrequency$, then this suggests another path exists between them. In this case, we would add a new edge $Age \rightarrow DevelopersMaturity$, as shown in Fig. 23(c).

4.2 Results

4.2.1 RQ1. What are the existing criteria to identify whether a software project uses CI?

Considering the Reichenbach's Common Cause Principle (PENROSE; PERCIVAL, 1962) and the Markov Condition (PEARL et al., 2000; HERNáN; ROBINS, 2010), in order to produce a sufficient causal DAG, we can exclude any variable that is not a common cause between CI and software quality or one of their ancestors. Thus, we remove from the DAG all non-common cause variables from the causal DAG in Fig. 19, for instance, $ResolutionTime$, and $ProgrammingLanguage$. In addition, we unified the variables $BugResolution$ and $BugReport$, as all connections and the meaning of $BugResolution$ are contained in $BugReport$.

Fig. 24 shows the literature-based DAG after the variable-selection process. According to the DAG, Continuous Integration (CI) has a direct influence on Bug Report (AMRIT; MEIJBERG, 2017b; VASILESCU et al., 2015), as well as an indirect influence through the Communication (CASSEE; VASILESCU; SEREBRENIK, 2020; HUANG; LIU; HUANG,) and developers' overconfidence variables (REBOUCAS et al., 2017; PINTO et al., 2018; HUANG; LIU; HUANG,). That means that CI influences the communication between contributors, which, in turn, communication influences the number of bug reports. A similar phenomenon occurs with developers' overconfidence, but overconfidence tends to have a negative effect on the number of bug reports, i.e., overconfidence may actually increase the number of bug reports or, at least, prevent a team from the opportunity of reducing the number of bug reports. Fig. 24 shows the literature-based DAG after the variable-selection process. According to the DAG, Continuous Integration (CI) has a direct influence on Bug Report (AMRIT; MEIJBERG, 2017b; VASILESCU et al., 2015), as well as an indirect influence through the Communication (CASSEE; VASILESCU; SEREBRENIK, 2020; HUANG; LIU; HUANG,) and developers' Overconfidence variables (REBOUCAS et al., 2017; PINTO et al., 2018; HUANG; LIU; HUANG,).

The age of a project influences the CI practices (ZAIDMAN et al., 2008; ZAIDMAN et al., 2011; HILTON; BELL; MARINOV, 2018; NERY; COSTA; KULESZA, 2019) and

also influences bug reports (VASILESCU et al., 2015), i.e., Age is a common cause for both CI and bug reports, thus opening a backdoor path for bias (see Section 2.4.2). Similarly, the commit frequency opens other backdoor paths: (i) $ContinuousIntegration \leftarrow CommitFrequency \leftarrow TestsVolume \rightarrow BugReport$; and (ii) $ContinuousIntegration \leftarrow CommitFrequency \leftarrow IssueType \rightarrow Communication \rightarrow BugReport$. These backdoor paths represent risks of confounding the influence of CI on other variables such as Bug Report. Thus, it is necessary to condition for some variables blocking these backdoor paths to obtain causal estimations regarding the total causal effect of CI on Bug Report. The minimal conditioning set is formed by *Age* plus $CommitFrequency$, capable of blocking all backdoor paths (i.e., confounding effects).

Conditioning for the confounding allows us to measure whether CI has a certain level of causal effect on Bug Reports despite the presence of confounding variables *Age* or $CommitFrequency$.

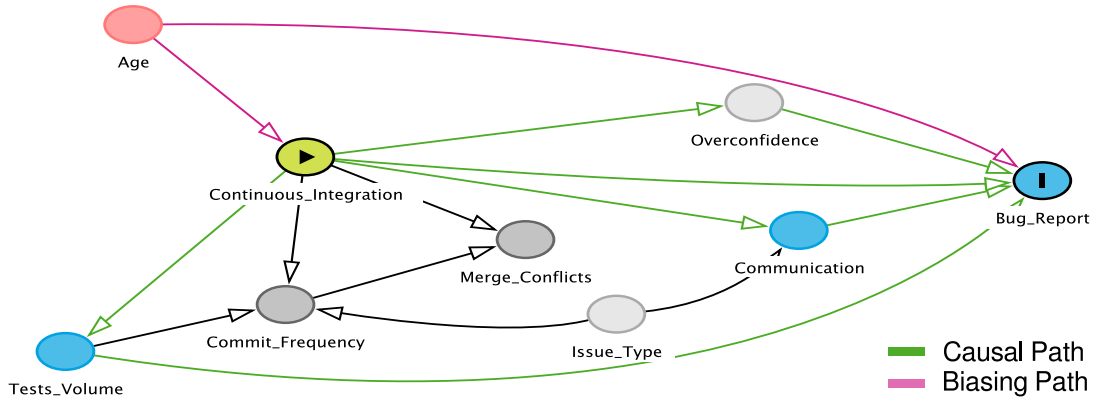


Figure 24 – Final literature-based DAG for CI, Bug Reports and their co-variables.

4.2.2 RQ2. What are the reported claims regarding the effects of CI on software development?

With the causal DAG obtained in RQ1 (Section 4.2.1) and the dataset we produced (Section 4.1.2.1), we can investigate if the data supports the relationships depicted in the causal DAG. Such evaluation is possible because we can derive statistically testable implications (see Section 2.4.1) using the d-separation properties on the causal DAG (see Fig. 24), and we can verify whether the same statistical independency also exists between the variables of the dataset. Thus, we answer RQ2 by testing the dataset against the proposed set of causal hypotheses from Table 29. To test such causal hypotheses, we use *dcov* test and *Kernel conditional independence test*.

When necessary, we test the unconditional independence hypotheses (i.e., two variables are independent without conditioning on other variables) using the *dcov* test. This test evaluates if two variables are independent, returning an R value between 0 and 1

when they are independent. If it returns a value greater than 1 the variables are considered to be dependent. For example, the *dcov* test for a hypothesis H_n "$Age \perp\!\!\!\perp CommitFrequency$" returns an R value of 49.5566 (i.e., larger than 1), meaning that we reject the hypothesis that Age and $CommitFrequency$ are independent.

On the other hand, we test the conditional independence hypotheses (i.e., two variables are independent when conditioning on another set of variables) in Table 29 using *Kernel conditional independence test* (HEINZE-DEML; PETERS; MEINSHAUSEN, 2018). This test evaluates the null hypothesis that two variables, Y and E, are independent conditioning on a set of variables X and returns a p-value for the null hypothesis. Thus, a small p-value (lower than the significance level) rejects the null hypothesis, indicating that the variables are not independent conditioning on a set of variables. If a p-value is high, the test fails to reject the null hypothesis, meaning that Y and E are likely independent conditioning on X.

As an example, the *Kernel conditional independence test* for H_m "$MergeConflicts \perp\!\!\!\perp TestsVolume \mid CommitFrequency$" (i.e., $MergeConflicts$ is independent of $TestsVolume$ conditioning on $CommitFrequency$) returns a p-value of 0.542159, failing to reject the null hypothesis for H_m. Thus, we assume that $MergeConflicts$ and $TestsVolume$ are independent when conditioning on $CommitFrequency$ since the p-value is higher than a significance level of 0.05 (in fact, we fail to reject the null hypothesis).

Table 29 – The results of the conditional independence tests for RQ2.

Conditional Independence	
Causal Hypotheses	**p-value**
H$_1$. $Age \perp\!\!\!\perp CommitFrequency \mid ContinuousIntegration$	**1.41e-07***
H$_2$. $Age \perp\!\!\!\perp TestsVolume \mid ContinuousIntegration$	**2.40e-12***
H$_3$. $Age \perp\!\!\!\perp Communication \mid ContinuousIntegration$	**1.16e-06***
H$_4$. $Age \perp\!\!\!\perp MergeConflicts \mid ContinuousIntegration$	0.254925
H$_5$. $BugReport \perp\!\!\!\perp CommitFrequency \mid Communication, ContinuousIntegration, TestsVolume$	**0.000000***
H$_6$. $BugReport \perp\!\!\!\perp MergeConflicts \mid CommitFrequency, ContinuousIntegration$	0.324586
H$_7$. $BugReport \perp\!\!\!\perp MergeConflicts \mid Communication, TestsVolume, ContinuousIntegration$	0.344638
H$_8$. $Communication \perp\!\!\!\perp MergeConflicts \mid CommitFrequency, ContinuousIntegration$	0.486532
H$_9$. $Communication \perp\!\!\!\perp TestsVolume \mid ContinuousIntegration$	**0.000000***
H$_{10}$. $MergeConflicts \perp\!\!\!\perp TestsVolume \mid CommitFrequency, ContinuousIntegration$	0.246583

$*p < 0.05$
$\perp\!\!\!\perp$ "Independent of..."
$\mid$ "Conditioning on..."

As we have seen in Table 29, we reject hypotheses H_1, H_2, H_3, H_5, and H_9. These

rejections are sufficient to conclude that if these d-separation implications from the causal DAG from our RQ1 (Fig. 24) are not true, the causal DAG is not consistent with the empirical data. That means that the actual development processes from our studied projects (which generate the observed data) differ from the relationships shown in the literature-based DAG (Fig. 24).

In simple terms, although our literature-based DAG indicates that *Age* and *CommitFrequency* are related only through *ContinuousIntegration* influence (i.e., they are independent when conditioning on *ContinuousIntegration*), our tests have shown that they actually share a different relationship (i.e., the tests did not confirm the conditional independence). Similarly, our tests show that the relationship between *Age* and the variables *TestsVolume*, *Communication*, and *MergeConflicts* are not independent when conditioning on *ContinuousIntegration*. Therefore, the tests suggest that *Age* potentially has a direct relationship with these variables.

A parallel case is the one concerning *Communication* and *TestsVolume*. The Literature-based DAG (Fig. 24) presents a relationship between these two variables passing through *ContinuousIntegration*. However, our tests did not confirm the conditional independence $Communication \perp\!\!\!\perp TestsVolume \mid ContinuousIntegration$.

Another example is the relationship between *BugReport* and *CommitFrequency*. Even if we disregard the influences of *Age*, *Communication*, *ContinuousIntegration*, and *TestsVolume*, they remain dependent, which means that our data shows that *BugReport* and *CommitFrequency* potentially have a direct relationship. Our data shows that *BugReport* potentially has a direct relationship with *CommitFrequency*. In RQ3 (Section 4.2.3), we deal with this set of observations on the failed hypothesis.

4.2.3 RQ3. Which empirical methods, projects and artifacts are used in the studies that investigate the effects of CI on software development?

To discover a plausible causal structure, we start from the failed causal hypotheses in RQ2 and explore where the literature-based DAG (Section 4.2.1) does not match the evidence from the empirical dataset (Section 4.2.2). We propose an alternative DAG (data-validated DAG) based on the observed mismatches and the testing hypotheses derived from such mismatches (i.e., failed causal hypotheses).

4.2.3.1 The relationship between *Age* and *CommitFrequency*

In the literature-based causal DAG in Fig. 24, there is a chain $Age \rightarrow ContinuousIntegration \rightarrow CommitFrequency$, as shown in Fig. 25 (a). Considering the d-separation rules (Section 2.4.1), we would expect that when conditioning on *ContinuousIntegration*, the association flow between *Age* and *CommitFrequency* would

interrupt, so they should be independent. However, in the analysis from RQ2, we rejected hypothesis H_1 ($Age \perp\!\!\!\perp CommitFrequency \mid ContinuousIntegration$, see Table 29).

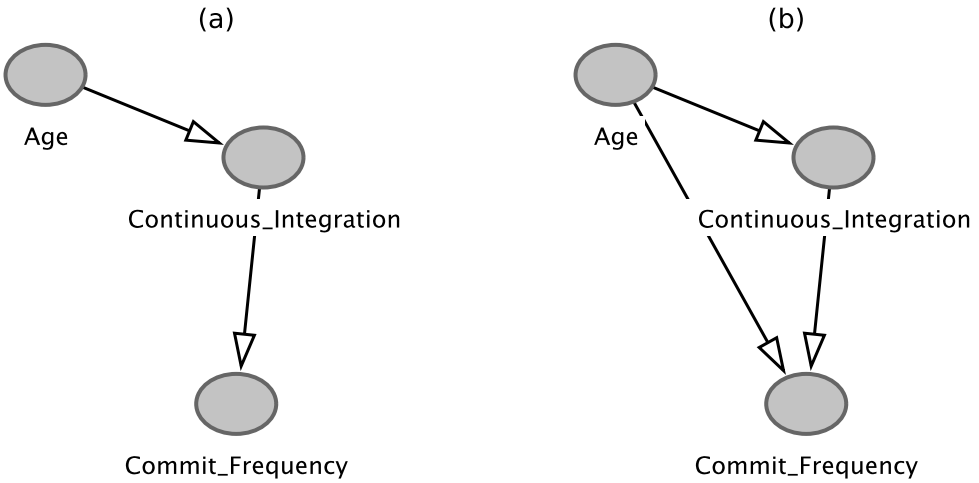


Figure 25 – (a) Causal structure (chain) involving Age, $ContinuousIntegration$, and $CommitFrequency$ as expressed in the literature-based causal DAG. (b) The new proposed causal structure concerning Age, $ContinuousIntegration$, and $CommitFrequency$ after statistical validations.

In this case, another path to the association between Age and $CommitFrequency$ may exist. An alternative hypothesis is a wrong collider structure between $Age \rightarrow BugReport \leftarrow Communication$. An association may occur through this path if this structure is not a collider. First, we tested the collider structure by formulating the hypothesis in Table 30. The hypothesis says that Age would become independent of $CommitFrequency$ when conditioning on $ContinuousIntegration$ and $BugReport$ ($Age \perp\!\!\!\perp CommitFrequency \mid ContinuousIntegration, BugReport$). The result rejects such a hypothesis, which means a persistent dependence between Age and $CommitFrequency$ when conditioning on $ContinuousIntegration$ and $BugReport$. Therefore, we add a new edge between Age and $CommitFrequency$, obtaining the structure shown in Fig. 25 (b).

Table 30 – Conditional independence test for the relationship between Age and $CommitFrequency$.

Conditional Independence	
Causal Hypotheses	**p-value**
H$_1$. $Age \perp\!\!\!\perp CommitFrequency \mid ContinuousIntegration, BugReport$	2.576492e−08*

$*p < 0.05 = dependence$
$\perp\!\!\!\perp$ "Independent of..."
$\mid$ "Conditioning on..."

4.2.3.2 The relationship between Age and $TestsVolume$

Analyzing hypothesis H_2 ($Age \perp\!\!\!\perp TestsVolume \mid ContinuousIntegration$, see Table 29) and the Fig. 26 (a), we observe that given the following chain: $Age \rightarrow$

$ContinuousIntegration \rightarrow TestsVolume$, there is an association between Age and $TestsVolume$, but this association would interrupt when conditioning on $ContinuousIntegration$. However, the test performed in RQ2 rejected such a hypothesis, revealing that Age and $TestsVolume$ remain dependent.

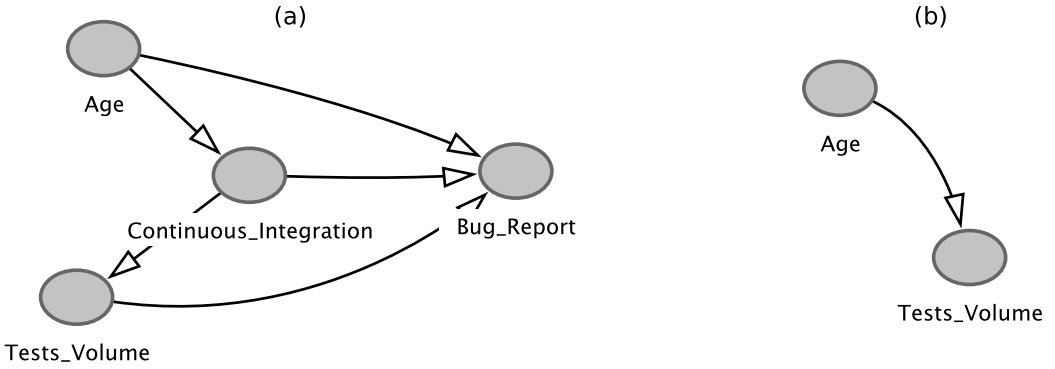


Figure 26 – (a) Causal structure involving Age and $TestsVolume$ as expressed in the literature-based causal DAG. (b) The new proposed direct path concerning Age and $TestsVolume$.

Table 31 – Conditional independence tests for the relationship between Age and $TestsVolume$.

Conditional Independence	
Causal Hypotheses	**p-value**
H$_1$. $Age \perp\!\!\!\perp TestsVolume \mid BugReport$	**0.0***
H$_2$. $Age \perp\!\!\!\perp TestsVolume \mid ContinuousIntegration, BugReport$	8.518427e−10*

$*p < 0.05$
$\perp\!\!\!\perp$ "Independent of..."
$\mid$ "Conditioning on..."

Therefore, these variables should have a direct association, or the structure that links them through $BugReport$ should be a chain rather than a collider. In this way, we should test this hypothesis in the form of conditional independence presented in Table 31, which expresses that if the structure is a chain when conditioning on the middle variable, Age, and $TestsVolume$ would become statistically independent. The test results in Table 31 show that even conditioning on $CI, and BugReport$, or just on $BugReport$, the variables Age and $TestsVolume$ remain dependent. Thus, we add a direct edge between them, as shown in Fig. 26 (b).

4.2.3.3 The relationship between *Age* and *Communication*

Based on the existing paths between Age and $Communication \mid ContinuousIntegration$ in the literature-based DAG (see Fig. 27 (a)), the hypothesis H_3 ($Age \perp\!\!\!\perp Communication \mid ContinuousIntegration$, see Table 29) says that Age and $Communication$ should be independent when conditioning on $ContinuousIntegration$.

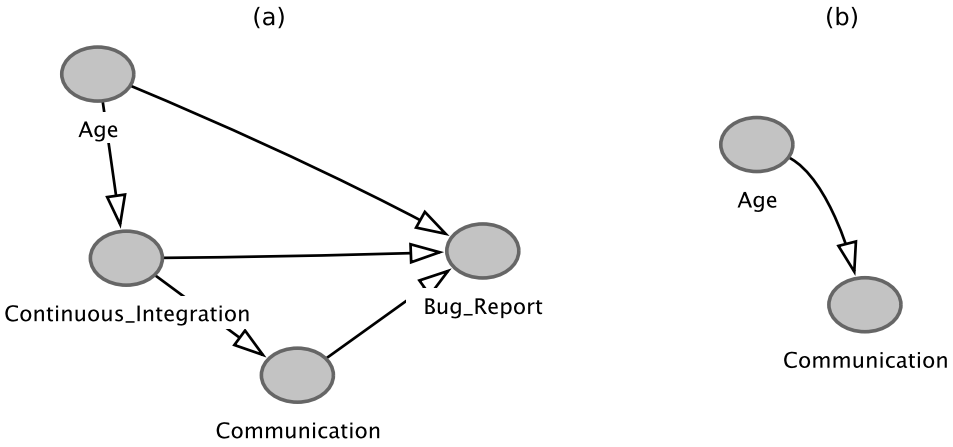


Figure 27 – (a) Causal paths between *Age* and *Communication* as expressed in the literature-based causal DAG. (b) The new proposed directed path between *Age* and *Communication*.

Rejecting hypothesis H_3 ($Age \perp\!\!\!\perp Communication \mid ContinuousIntegration$) indicates that *Age* and *Communication* remain dependent. This dependency may be due to a wrong structure in the collider $Age \rightarrow BugReport \leftarrow Communication$ that could be a chain, but in this case, when conditioning in *BugReport*, *Age*, and *Communication* would become independent.

Another hypothesis is a wrong structure in the chain $Age \rightarrow ContinuousIntegration \rightarrow Communication$ that may be a collider, and then when we conditioned in $ContinuousIntegration$, we opened the association flow, making *Age* and *Communication* dependent. In this way, we should test these hypotheses by testing the independencies depicted in Table 32 and Table 33.

Table 32 – Conditional independence tests for the hypothesis related to relationship between *Age* and *Communication*.

Conditional Independence	
Causal Hypotheses	**p-value**
H_1. $Age \perp\!\!\!\perp Communication \mid BugReport$	**0.0***
H_2. $Age \perp\!\!\!\perp Communication \mid BugReport, ContinuousIntegration$	**1.251478e-05***

$*p < 0.05$
$\perp\!\!\!\perp$ "Independent of..."
$\mid$ "Conditioning on..."

Table 33 – Unconditional independence tests for the hypothesis related to relationship between *Age* and *Communication*.

Unconditional Independence	
Causal Hypotheses	**R**
H_3. $Age \perp\!\!\!\perp Communication$	**11.86268 ***

$*R > 1$
$\perp\!\!\!\perp$ "Independent of..."

The test result in Table 32 shows that even conditioning on $BugReport$, the variables Age and $Communication$ remain dependent. Thus, there exists a dependency through another path. The result in Table 33 does not confirm the hypothesis of a collider in $Age \rightarrow ContinuousIntegration \leftarrow Communication$ since without conditioning on $ContinuousIntegration$, Age and $Communication$ remain dependent. Therefore, we add an edge $Age \rightarrow Communication$, as illustrated in Fig. 27 (b).

4.2.3.4 The relationship between $CommitFrequency$ and $BugReport$

The rejection of hypothesis H_5 ($BugReport \perp\!\!\!\perp CommitFrequency \mid Communication$, $ContinuousIntegration$, $TestsVolume$, see Table 29), similarly, indicates that the structures shown in Fig. 28 (a) are not accurate.

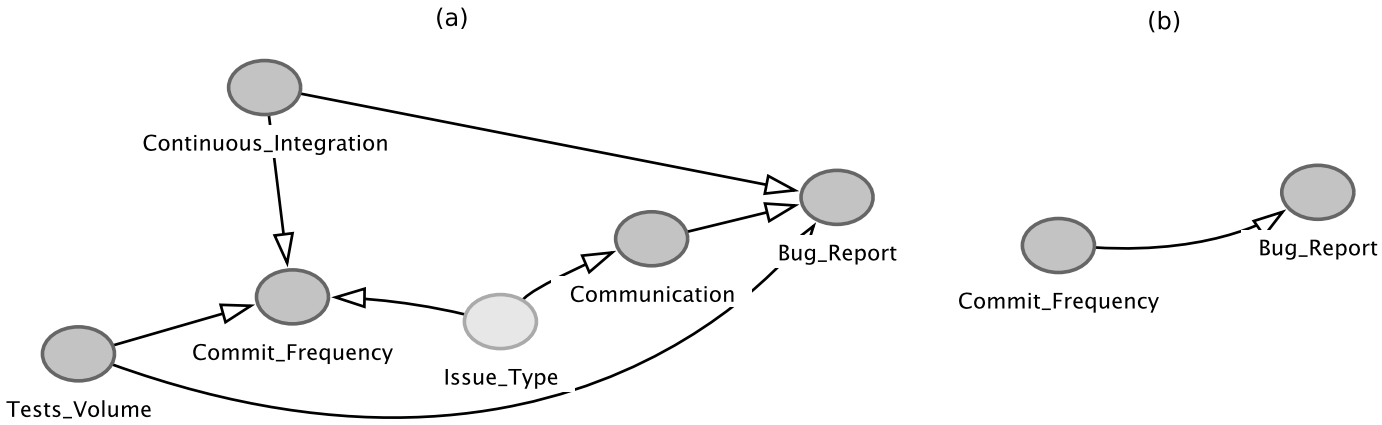


Figure 28 – (a) Causal paths between $CommitFrequency$ and $BugReports$ as expressed in the literature-based causal DAG. (b) The new proposed directed path between $CommitFrequency$ and $BugReports$.

According to these structures, we expected that $BugReport$ and $CommitFrequency$ would become independent when conditioning on $ContinuousIntegration$, $Communication$, and $TestsVolume$. However, the test did not confirm the independence (see Table 29), even testing all variables conditioned together or separately. On the other hand, we tested if $BugReport$ and $CommitFrequency$ are unconditionally independent and got an $R-value$ of 6.009902, indicating a dependency between them.

These tests suggest the existence of another path between $BugReport$ and $CommitFrequency$. Thus, we add a new edge linking them as shown in Fig. 28 (b).

4.2.3.5 The relationship between $Communication$ and $TestsVolume$

Analyzing the rejection of hypothesis H_9, we observed that when conditioning $Communication$ and $TestsVolume$ on $ContinuousIntegration$ (see Fig. 29 (a)), the dependence between $Communication$ and $TestsVolume$ does not disappear. We then tested if the collider $TestsVolume \rightarrow CommitFrequency \leftarrow IssueType$ would be wrong.

If this hypothesis is correct, the association is flowing through this path, then conditioning on $CommitFrequency$ would block the association.

In this sense, we test the conditional independence presented in Table 34 and confirm the hypothesis that the structure is not a collider. Therefore, we reorient the edge between $CommitFrequency$ and $IssueType$, as present Fig. 29 (b).

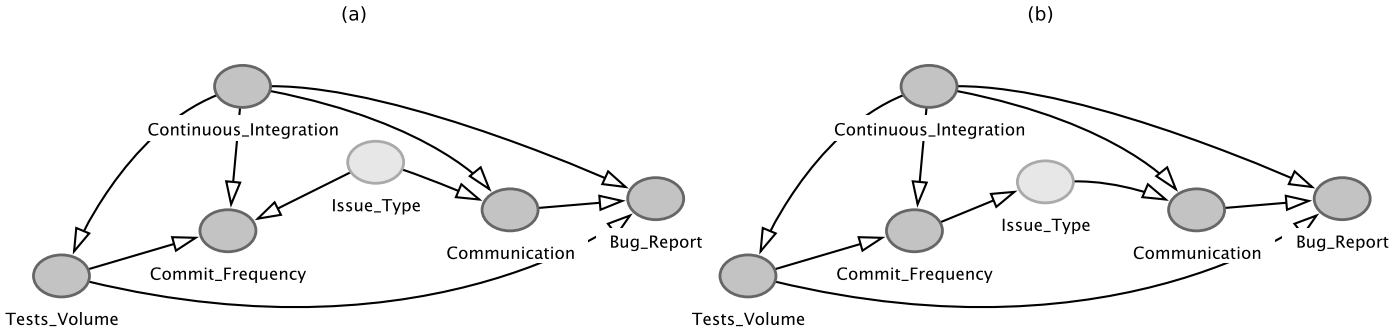


Figure 29 – (a) Causal structure between $Communication$, and $TestsVolume$ as expressed in the literature-based causal DAG. (b) The new proposed structure between $Communication$ and $TestsVolume$ after statistical validations. The edge between $CommitFrequency$ and $IssueType$ was inverted.

Table 34 – Conditional independence tests for the hypothesis related to relationship between $Communication$ and $TestsVolume$.

Conditional Independence	
Causal Hypotheses	**p-value**
$\mathbf{H_1}$. $Communication \perp\!\!\!\perp TestsVolume \mid ContinuousIntegration, CommitFrequency$	0.862325

$*p < 0.05$
$\perp\!\!\!\perp$ "Independent of..."
$\mid$ "Conditioning on..."

4.2.3.6 Data-Validated Causal DAG

Afterward, we obtain the first version of data-validated DAG in Fig. 30, raising a new set of statistical implications to test. Table 35 shows such implications as causal hypotheses and the results of the independence tests.

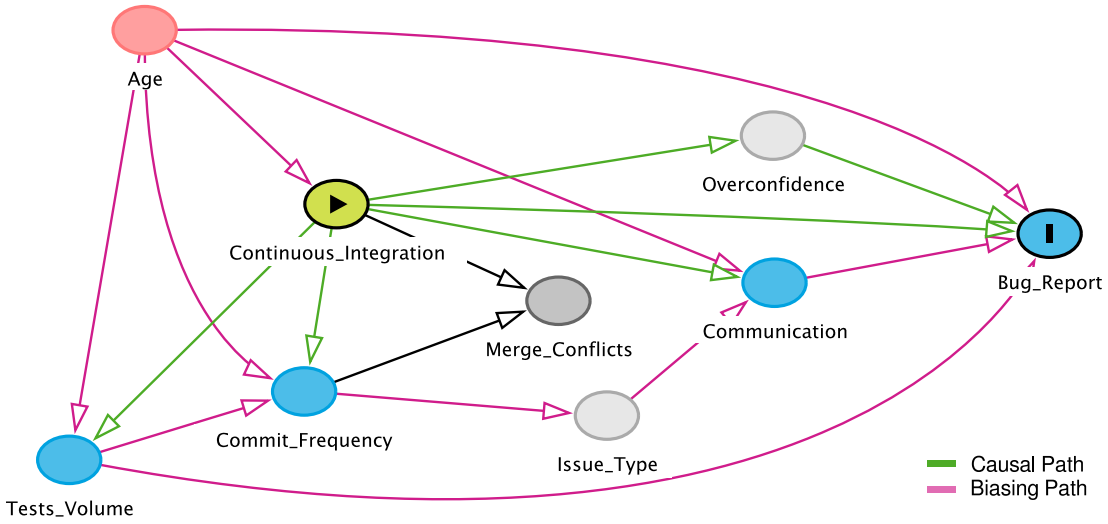


Figure 30 – Initial version of data-validated DAG for CI, Bug Reports and their co-variables.

Table 35 – The results of the conditional independence tests for RQ3.

Conditional Independence	
Causal Hypotheses	**p-value**
H$_1$. $Age \perp\!\!\!\perp MergeConflicts \mid CommitFrequency, ContinuousIntegration$	0.269635
H$_2$. $BugReport \perp\!\!\!\perp MergeConflicts \mid CommitFrequency, ContinuousIntegration$	0.321338
H$_3$. $Communication \perp\!\!\!\perp MergeConflicts \mid CommitFrequency, ContinuousIntegration$	0.486532
H$_4$. $Communication \perp\!\!\!\perp TestsVolume \mid Age, CommitFrequency, ContinuousIntegration$	**2.05e-12 ***
H$_5$. $MergeConflicts \perp\!\!\!\perp TestsVolume \mid CommitFrequency, ContinuousIntegration$	0.246583

*$p < 0.05$
$\perp\!\!\!\perp$ "Independent of..."
$\mid$ "Conditioning on..."

After the statistical analysis, the H_4 ($Communication \perp\!\!\!\perp TestsVolume \mid Age, CommitFrequency, ContinuousIntegration$, see Table 35) was rejected. This result means that even conditioning on $Age, CommitFrequency$ and $ContinuousIntegration$, the variables $Communication$ and $TestsVolume$ remain dependent, i.e., another open path exists between them. Since the paths by $BugReport$ is a collider (see Fig. 31 (a)), we may infer that there is a path that this first version of data-validated DAG (Fig. 30) does not reflect the true causal relationship. Therefore, we add a new edge between $TestsVolume$ and $Communication$ (as shown in Fig. 31 (b)) and achieve a new version of the data-validated DAG shown in Fig. 32.

The data-validated DAG shown in Fig. 32 raises a new set of statistical implications depicted in Table 36. After the statistical analysis, all these four hypotheses were confirmed,

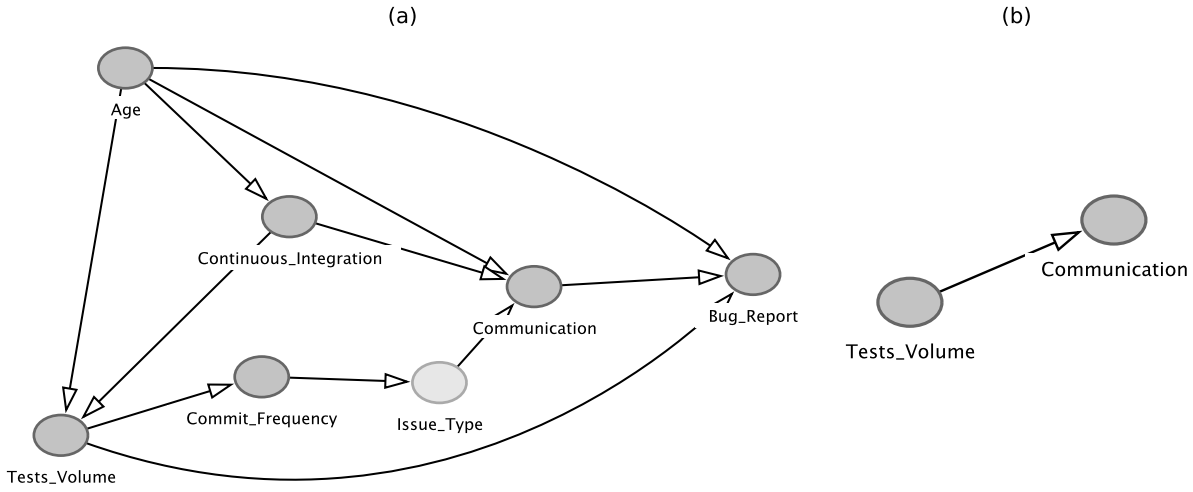


Figure 31 – (a) Causal paths between $TestsVolume$ and $Communication$ as expressed in the literature-based causal DAG. (b) The new proposed direct path between $TestsVolume$ and $Communication$.

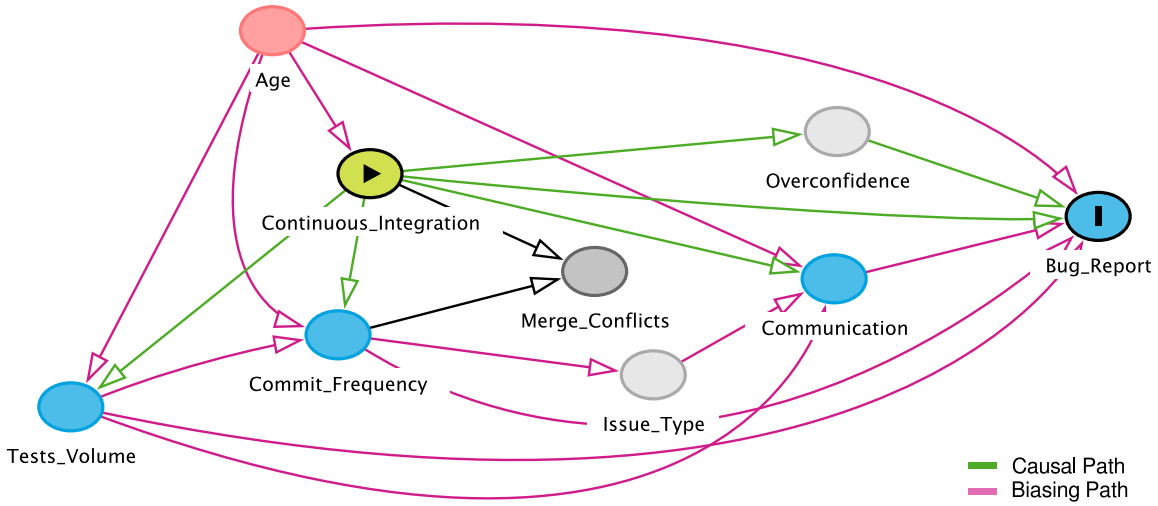


Figure 32 – data-validated DAG for CI, Bug Reports and their co-variables.

meaning that the dataset upholds all implications of the data-validated DAG. Thus, we can conclude that we find a plausible model in our data-validated DAG (Fig. 32).

Table 36 – The results of the conditional independence tests for RQ3.

Conditional Independence	
Causal Hypotheses	**p-value**
H$_1$. $Age \perp\!\!\!\perp MergeConflicts \mid CommitFrequency, ContinuousIntegration$	0.269635
H$_2$. $BugReport \perp\!\!\!\perp MergeConflicts \mid CommitFrequency, ContinuousIntegration$	0.321338
H$_3$. $Communication \perp\!\!\!\perp MergeConflicts \mid CommitFrequency, ContinuousIntegration$	0.486532
H$_4$. $MergeConflicts \perp\!\!\!\perp TestsVolume \mid CommitFrequency, ContinuousIntegration$	0.384515

$^*p < 0.05$
$\perp\!\!\!\perp$ "Independent of..."
$\mid$ "Conditioning on..."

4.3 Discussion

In RQ1 4.2.1, we cataloged a set of claims related to Continuous Integration that we verified empirically in RQ2 4.2.2. Finally, in the RQ3 4.2.3, we proposed a new causal DAG mixing the literature knowledge with the empirical data.

The data-validated DAG shown in Fig. 32 shows a more significant influence of *Age* than in the previous literature-based DAG (Fig. 24). *Age* is an important source of confounding effects since it relates to $TestsVolume$, $CommitFrequency$, $ContinuousIntegration$, $Communication$, and $BugReport$.

In the data-validated DAG, we can also observe that $ContinuousIntegration$ influences $CommitFrequency$ and $MergeConflicts$, while $CommitFrequency$ contributes to more $MergeConflicts$ and indirectly to more *communication*. This way, $ContinuousIntegration$ influences, directly and indirectly, to obtain more $Communication$ in the software development process. $Communication$, in turn, has a causal relationship with $BugReport$.

The negative side of $ContinuousIntegration$ is the occurrence of $MergeConflicts$ and the promotion of an $Overconfidence$ in the development environment. We conjecture that CI is believed to decrease the commit size because of the increased commit frequency. This higher frequency engenders a favorable scenario for an increase in merge conflicts.

As for the overconfidence, it may arise from the belief that the automated environment provided by CI is always reliable. Thus, we refer to it as confidence over the reasonable, which is not necessarily grounded on objective mechanisms and metrics that give such security and confidence(e.g., code coverage, test quality, code smells checking). Such an $Overconfidence$ may increase $BugReport$ since developers may relax their attention to detail, outsourcing such quality control to automation developed in the scope of CI.

Finally, we observe that $ContinuousIntegration$ also, directly and indirectly, influences $BugReport$. $ContinuousIntegration$ has a direct relationship with $BugReport$ and

has an indirect influence through $Communication$, $CommitFrequency$, $TestsVolume$, and $Overconfidence$. While $Communication$, $CommitFrequency$, and $TestsVolume$ affects $BugReport$ negatively, $Overconfidence$ affects $BugReport$ positively.

4.3.1 Implications for researchers

The DAGs presented in this study, especially the data-validated DAG (Fig. 32), may serve as a baseline for further CI and software quality investigations. This DAG brings findings from the combination of assumptions from the literature confronted with an empirical dataset. Other studies can reinforce, refute, or expand our findings and the DAGs, contributing to a more robust theory on the effects of CI on software quality (our replication package[12] may be helpful).

We can also consider the differences emerging from the comparison between the literature-based DAG (Fig. 24) and the data-validated DAG (Fig. 32). The changed associations arise from the data and reveal exciting findings to be further investigated by researchers. As well as the edges additions originated in data-validated DAG.

The data-validated DAG (Fig. 32) added new edges. The addition of $Age \rightarrow CommitFrequency$ (Section 4.2.3.1) aligns with Zhao et al. (ZHAO et al., 2017) that found a slight decreasing trend in the number of non-merge commits over time. As interpreted by Brindescu et al. (BRINDESCU et al., 2014), this finding could be related to the project activity level changes during their lifecycle. The projects tend to switch activities from adding new features to performing corrective changes and similar ones (BRINDESCU et al., 2014).

In this line, as the project ages, more $TestsVolume$ tend to exist ($Age \rightarrow TestVolume$). Despite the literature used to build literature-based DAG in our RQ1 (Section 4.2.1) does not explore the direct relationship between Age and $TestsVolume$, our dataset analysis reveals it. Sizilio Nery et al. (NERY; COSTA; KULESZA, 2019) investigates the evolution of test ratio (a metric similar to our $TestsVolume$) and relates a change in the test ratio evolution trend to CI employment, NOCI projects have a negligible change in the test ratio over time. In our dataset, a causal relationship exists between Age and $TestsVolume$, i.e., as the Age of a project advances, the greater the $TestsVolume$ independently of CI employment or not. Future investigations may study qualitatively how the tests evolve regarding the quality and motivation behind such test efforts.

So is the Age influence on $CommitFrequency$ and $TestsVolume$, Age also causes an increase in $Communication$, in terms of pull requests comments. For example, an interesting observation arises from the causal flow $Age \rightarrow Communication$ of our data-validated DAG. We note that in the literature, Cassee et al. [46] investigate the association between CI adoption and pull-request communication. Our study, however, identifies an effect on communication separated from CI, indicating that the Age of a project may

[12] https://github.com/elieziosoares/ci_quality_study_replication

affect *Communication* over time rather than CI. Therefore, our data-validated DAG can be used to understand better the existing associations observed in the literature.

In this line of thought, we demonstrate in Section 4.2.3.6 that $TestsVolume$ also influences *Communication* ($TestsVolume \rightarrow Communication$). It is possible that as test volume increases, more discussion is generated regarding the commits. There is an interrelationship between $TestsVolume$, $CommitFrequency$, and *Communication*, as discussed by Fowler: "Integration is about communication" (MARTIN; MATTHEW, 2006).

Similarly, these three variables affect *BugReport*. Section 4.2.3.4 presents an addition of the edge $CommitFrequency \rightarrow BugReport$, suggesting that the more frequent the commits, the less the *BugReports*. Indeed, the more integration, the more builds and quality checks are performed. Eyolfson et al. (EYOLFSON; TAN; LAM, 2014) found that more frequent committers produce fewer bug-introducing commits. $CommitFrequency$ could have a similar foundation: the more frequent the commits, the more knowledge, and familiarity with the domain, architecture, and tools used in the project, providing less faulty development. Researchers could empirically study the relationship between commit frequency and code or project quality.

If more tests and frequent commits mean more communication, then CI essentially promotes more communication, directly and indirectly. Moreover, more communication generates more knowledge and fewer errors, thus producing a high-quality software product. Cassee et al. (CASSEE; VASILESCU; SEREBRENIK, 2020) highlight the necessity of investigating the social aspects of software development. In their work, Cassee et al. (CASSEE; VASILESCU; SEREBRENIK, 2020) relate CI with a decrease in the number of comments in code reviews; however, they call CI a "Silent Helper" because CI plays a role communicating sufficiently to maintain the same level of activity of projects with more comments. This view corroborates the idea that the CI environment, $CommitFrequency$, $TestsVolume$, all of it is about communication.

Further studies may investigate this techno-social aspect, answering how and why $CommitFrequency$ and $TestsVolume$ can potentialize communication. Furthermore, investigating how the CI environment contributes to communication, i.e., in which aspects it communicates and the mechanisms, tools, and metrics through which CI is manifested, may mitigate the harmful false sense of confidence ($Overconfidence$) generated by CI in some environments. For instance, Bernardo et al. (BERNARDO et al., 2023) highlight CI influencing higher confidence during code review.

In addition, future studies may analyze whether CI is the cause or the consequence of attracting or attaining more experienced (core) developers. Suppose a clear causal flow exists between $ExperiencedDevelopers \rightarrow CI$ or $CI \rightarrow ExperiencedDevelopers$. In that case, other potential investigations will open, such as investigating whether CI generates more or less bug reports because, for instance, experienced developers detect more bugs.

The data-validated causal DAG (Fig. 32) shows *Age* as a significant source of

confounding (constituting a backdoor path) for $ContinuousIntegration$ and $BugReport$. Thus, researchers should be more attentive to controlling for Age when studying the effects of CI. Additionally, researchers may investigate the rationale behind this relationship between older projects and CI projects. They may explore, for example, the decision process for CI adoption and the challenges involved in such a process that could postpone the adoption.

4.3.2 Practical implications

Our study and the data-validated DAG (Fig. 32) raise practical implications. Software developers may understand that they potentially decrease the bug reports when employing CI in their projects. CI causes an effect on the $BugReport$, encourages team $Communication$, and increases the $CommitFrequency$ (in turn, both positively affect bug reports). Therefore, considering the observed literature-reported relationships and the variables we were able to measure in our study, CI has a causal effect on the number of Bug Reports, which, in turn, is our proxy for software quality.

On the other hand, CI projects have greater values for the Age variable. Adopting CI may be related to greater project maturity. This demand may be related to the difficulties of implementing CI and a demand for more qualified and experienced professionals. Awareness of (over)confidence in a flaky automated environment is also essential. Investing in building a reliable environment could mitigate the negative overconfidence effect.

4.4 Threats To Validity

In this section, we discuss the threats to the validity of our study.

Construct Validity. There are construct threats associated with the search for associations between CI and software quality from existing software engineering literature. These associations are used to build the DAG, showing how CI may influence software quality with potential confounding variables (RQ1). To reduce the threats associated with the search for such associations, we have mainly used existing systematic literature reviews and empirical studies to search for research work associated with CI and software quality. However, there are always associations (open questions) that are not captured by existing studies or even studies that were not found and considered in our analysis. In addition, we have also considered the associations related to CI sub-practices present in the selected papers as a direct relationship between CI and other important software development aspects.

We also have construct threats associated with the repository mining of the collected data that supports the statistical tests of the associations in the DAG (RQ2). We used GitHub and Travis CI APIs to collect data about the selected projects and compute the metrics. Any bias in how we compute such metrics can affect our results. For example,

we can measure communication in other ways, such as calls, meetings, emails, etc. As another example, merge conflicts and bug issues were cataloged based on heuristics that could carry biases preventing our scripts from precisely identifying all issues and conflicts. However, we did our best to collect the seven considered metrics in this study. We seek to follow methods and metrics already known in the software engineering community, as we show throughout Section 4.1.

Internal Validity. To compute the metrics associated with our dataset, we aggregate them into months. Such a strategy can produce a lack of information (compared to a week or day-based analysis) associated with the metrics of interest. Recent empirical studies (NERY; COSTA; KULESZA, 2019; BERNARDO; COSTA; KULESZA, 2018) that investigate the impact of CI have adopted a similar month-based strategy.

External Validity. External threats concern the extent to which we can generalize our results. Our work analyzed a total of 59,034 commits, 16,619 builds, and 3,199 bug reports of 148 Github open-source projects - 74 CI and 74 no-CI projects. Nevertheless, we collected a consistent dataset representative of the open-source community, which validates our DAG. However, other studies would be needed to reinforce, refute, or modify our causal DAG. The more projects and metrics are reliable and consistently collected, the more robust the causal DAG and the causal theory proposed would be. This study is not intended to be definitive concerning the causal relationship between CI and software quality but rather to be a starting point in such a direction. More studies are needed to evolve our understanding of causal relationships between CI and quality.

4.5 Conclusion

Continuous Integration (CI) has increased in popularity in the last decades with the expectation of providing several software development benefits. One of these benefits associated with CI is increased software quality. This study expands the current software engineering body of knowledge by investigating the association between CI and software quality from a causal perspective. We reviewed the literature cataloging relevant CI and software quality associations and adjacent relationships. Then, we built the literature-based causal DAG (RQ1) expressing these associations. We mined software repositories to empirically assess the literature-based causal DAG (RQ2), analyzing releases in 12 activity months from 148 software projects.

Analyzing the testable implications from the literature-based causal DAG generated from literature assumptions, we found in RQ2 some associations that do not exist in the collected dataset. Then, relying on a hybrid literature-data approach, we proposed a new data-validated causal DAG (RQ3), expressing the relationship between the relevant variables.

We discussed research opportunities from the differences cataloged between the

literature-based DAG and data-validated causal DAG. Researchers and practitioners can benefit from our findings based on the proposed data-validated causal DAG that, in summary, shows a positive causal effect from CI on bug reports. CI also influences developers' communication, reinforcing bug report benefits. The developers' overconfidence is still a concern in CI environments and could negatively affect bug reports.

5 Related Work

This chapter positions this thesis, relating it with the main related work. We discuss other systematic literature reviews related to Continuous Integration in Section 5.1 and studies addressing CI and software quality in Section 5.2.

5.1 Systematic Literature Reviews in CI

This section discusses *systematic literature reviews* (SLR) related to our work. We highlight the main differences in contributions and findings of five other SLRs, as shown in Table 37. In particular, Dikert K et al. studied agile methods (DIKERT; PAASIVAARA; LASSENIUS, 2016), Laukkanen E et al. studied continuous delivery (LAUKKANEN; ITKONEN; LASSENIUS, 2017), Shahin et al. studied continuous integration, delivery and deployment (SHAHIN; BABAR; ZHU, 2017). Two other studies by Ståhl & Bosch investigated the existing literature regarding CI (STåHL; BOSCH, 2013; STåHL; BOSCH, 2014b).

Ståhl & Bosch (STåHL; BOSCH, 2013) investigated which known benefits of CI are experienced in the industry. They conducted a systematic literature review, including 33 articles with 7 explicit claims regarding the benefits of CI. They interviewed 22 individuals (developers, testers, project managers, and line managers) from 4 projects to complement the study. Their results reveal high standard deviations in answers, indicating disparate reported experiences.

Another study by Ståhl & Bosch (STåHL; BOSCH, 2014b), motivated by their previous work, performed a literature review on CI to understand the different benefits of CI adoption better. The new study included 46 articles to find differing practices, supporting identifying potential CI variation points. They synthesized the extracted statements in 22 clusters, of which only six do not have disagreements. In addition, the study proposes a descriptive model for documenting these variations. It highlights the need to better document such CI variants to better understand their benefits or disadvantages.

Dikert et al. (DIKERT; PAASIVAARA; LASSENIUS, 2016), conducted an SLR on large-scale agile transformations (i.e., changes in practices or organizational culture in companies with 50 or more people, or at least six teams) to identify success and challenge factors. They searched for papers describing industrial cases in agile development adoption, including 52 publications in a thematic synthesis. The authors documented 35 challenges in 9 categories and 29 success factors distributed into 11 categories.

Shahin et al. (SHAHIN; BABAR; ZHU, 2017) studied 69 papers in a SLR to classify approaches/tools and to identify challenges and practices in Continuous Integration, Continuous Delivery, and Continuous Deployment. The study's contributions include

classifying approaches/tools, a list of critical factors to implement continuous practices, a guide to select approaches/tools, and a list of research directions.

Laukkanen et al. (LAUKKANEN; ITKONEN; LASSENIUS, 2017), also performed a SLR to explore the reported problems when adopting Continuous Delivery. Their study also identified causes and solutions to these problems. The study selected 30 articles that found 40 problems and 29 solutions. The problems and solutions were classified into seven themes, e.g., integration, testing, and building design.

Some of these studies focus on a more general perspective, such as Dikert et al. (DIKERT; PAASIVAARA; LASSENIUS, 2016) which focused on agile methods adoption, and Shahin et al. (SHAHIN; BABAR; ZHU, 2017) which studied continuous integration, delivery, and deployment. Laukkanen et al. (LAUKKANEN; ITKONEN; LASSENIUS, 2017) studied continuous delivery, which differs from the implications of continuous integration. Our work focuses strictly on continuous integration, and we explore studies in-depth, analyzing and comparing their findings.

Ståhl & Bosch (STåHL; BOSCH, 2013) also perform analyses strictly related to CI investigating the experienced benefits in the industry. However, while our study exhaustively explores the literature to analyze the claims related to CI (benefits and cons), Ståhl & Bosch (STåHL; BOSCH, 2013) do not provide an exhaustive list of CI benefits nor explore the potential adverse effects (or challenges) of adopting CI. In their subsequent work, Ståhl & Bosch (STåHL; BOSCH, 2014b) cataloged CI variation points. Our systematic literature review complements their work because we discuss the empirical claims related to CI across six different themes (each representing an area of software development). Our work helps practitioners and researchers to obtain a holistic view of the implications of using continuous integration in different areas of software development and different granularity of CI practices.

Unlike the studies mentioned above, our SLR (Chapter 3) analyzes a substantially larger sample of articles, i.e., 101 studies ranging from 2003 to 2019. Considering the years of the related research presented, it is noticeable that the newest (i.e. Laukkanen et al. (LAUKKANEN; ITKONEN; LASSENIUS, 2017) and Shahin et al. (SHAHIN; BABAR; ZHU, 2017)) is dated to 2017, including mostly primary from 2016 (SHAHIN; BABAR; ZHU, 2017). Study 1 contributes to the community by advancing at least three years in the related literature. Our study also innovates by studying the effects of CI and considering the specific CI practices within different CI settings. Moreover, our work also analyzes the methodologies of our 101 selected studies.

5.2 Software Quality in CI

Zaytsev & Morrison demonstrated in a case study with mining software repository (MSR) that CI contributes to reducing the turnaround time for resolving broken

Table 37 – Systematic Literature Reviews (SLRs) that related to our work. We show the authors, focus, findings, number of included articles, and the year of publication.

Study	Focus	Findings	# Papers	Year
Ståhl and Bosh (STåHL; BOSCH, 2013)	Continuous integration	Benefits of CI	33	2013
Ståhl and Bosh (STåHL; BOSCH, 2014b)	Continuous integration	Differences in CI practices	46	2014
Dikert et al. (DIKERT; PAASIVAARA; LASSENIUS, 2016)	Large-scale agile transformations	Challenges and success of agile adoption	52	2016
Shahin et al. (SHAHIN; BABAR; ZHU, 2017)	Continuous integration, delivery, and deployment	Approaches, tools, challenges and practices	69	2017
Laukkanen et al. (LAUKKANEN; ITKONEN; LASSENIUS, 2017)	Continuous delivery	Problems, causes and solutions.	30	2017

builds (ZAYTSEV; MORRISON, 2013). In a mining software repository investigation using large, historical data on process metrics and outcomes in GitHub projects, Vasilescu et al. conclude that CI improves teams' productivity without compromising code quality. In this way, they observe that the core developers can discover significantly more bugs when using CI than in teams not using CI (VASILESCU et al., 2015).

Hilton and colleagues conducted a mixed-method study (MSR / survey) and found CI associations with earlier bug catching. Teams using CI feel less worried about breaking builds, and despite less effort to reject problematic pull requests, they spend less time debugging (HILTON et al., 2016). In another mixed-method study (survey/interviews) (HILTON et al., 2017), they confirmed such findings and concluded that projects with CI give more value to automated tests, promoting high-quality code and tests. These conclusions are supported by a common build environment and test workflows running through all platforms.

Amrit and Meijberg conducted a case study combining CI and Test-Driven Development (TDD) (AMRIT; MEIJBERG, 2017a). They found that projects presented fewer post-release defects when using CI, finding and fixing them before releases. Rahman et al. analyzed more than 250 open-source and proprietary projects and remarked that CI increases the number of issues and bugs resolved (RAHMAN et al., 2018). Another case study (KAYNAK; ÇILDEN; AYDIN, 2019) also highlighted CI related to earlier bug catching and less time debugging.

Pinto et al. conducted a survey study with 158 CI users in open-source projects (PINTO et al., 2018). They recorded their perceptions about the reasons for build breakage and the benefits and problems associated with CI systems. Among other findings, they associated CI with improved software quality, support to catch problems earlier, enforcement of automated software testing, and enabling cross-platform testing. On the other hand, they remarked an association with a false sense of confidence when the tests are insufficient or flaky.

While the presented studies are grounded in association relationships, our study presents a causal investigation between CI and software quality. For this purpose, we rely on previous literature assumptions, a mining software repository study, and the employment of the causal DAGs technique. In this sense, our study innovates in offering seminal empirical causal conclusions regarding CI and software quality.

6 Conclusions

Continuous Integration (CI) has increased in popularity and attracted the attention of practitioners and researchers to its effects on software development. Several studies have empirically investigated such effects, and we explored and cataloged them through Study 1 (Chapter 3). The reported benefits of CI are increased software quality and improved team productivity. Existing studies present CI associated with defect reduction and decreased time to address defects (VASILESCU et al., 2015; AMRIT; MEIJBERG, 2017a; KAYNAK; ÇILDEN; AYDIN, 2019). Studies also present CI associated with increased productivity and efficiency (STåHL; BOSCH, 2013; VASILESCU et al., 2015; HILTON et al., 2016; HILTON et al., 2017; KAYNAK; ÇILDEN; AYDIN, 2019).

Despite the value and diversity of these existing studies, they are correlation studies and do not provide reliable conclusions regarding whether CI causes such effects. This thesis empirically expands the current knowledge by investigating the relationship between CI adoption and software quality and CI adoption and team productivity from a causal perspective. We present the contributions of this thesis next.

6.1 Contributions and Findings

This thesis aims to understand the effects of Continuous Integration (CI) on software development by identifying the existing reported outcomes and aiming to advance the knowledge by empirically investigating the causal relationships between them. Specifically, we performed causal studies on the relationship between CI and software quality and CI and team productivity. To achieve this goal, we conducted a systematic literature review, a causal explanatory study on CI and software quality, and a causal explanatory study on CI and team productivity.

The main findings are highlighted below:

6.1.1 Study 1: What are the reported effects of CI on software development? (Chapter 3)

- Findings about the benefits of CI usage:
  - Although CI may incur technical challenges to the team (e.g., creating a reliable automated build environment), CI is mentioned as a success factor in software projects (Section 3.2.2.2).
  - We find evidence for the association between CI and improved productivity, efficiency, and developer confidence (Sections 3.2.2.1 and 3.3.2.2).

  - Continuous integration benefits the *software process* by promoting faster iterations, more stability, predictability, and transparency in the development process (Section 3.2.2.2).
  - CI can also benefit the pull-based development by improving and accelerating the integration process (Section 3.2.2.4).
  - Our study indicates that CI positively influences the way developers perform commits (Section 3.2.2.4).
  - The studies demonstrate a perceived provision of transparency and continuous quality inspections when CI is adopted (Section 3.2.2.3).
  - There is evidence of the association between CI and better testing (Section 3.2.2.3).
  - We find that studies credit CI to an improvement in the time to find and fix issues and a decrease in defects reported (Section 3.2.2.5).
  - The studies reveal that CI impacts the build process, promoting good practices related to build health and contributing to an increase in successful builds (Section 3.2.2.6).

- Findings about the drawbacks of CI usage:
  - CI may introduce complexity to the project, requiring more effort and discipline from developers, negatively impacting developers' perceived productivity (Sections 3.2.2.1 and 3.3.2.2).
  - Some studies also discuss the false sense of confidence, i.e., when developers blindly rely on flaky tests (Sections 3.2.2.1 and 3.3.2.2).
  - There are studies reporting that CI may prolong the pull request lifetime (Section 3.2.2.4).

- Findings about the primary studies:
  - The study shows that 42.5% of the primary studies did not present explicit criteria to identify projects that use CI (Section 3.2.1).
  - We also found that 15.8% used only one criterion (e.g., more than half of studies used "automated builds" as a criterion).
  - Regarding the criteria applied to check whether participants use CI or not (Sections 3.2.1 and 3.3.1), our findings reveal the need for performing other checks during interviews or surveys related to the adherence of CI beyond the self-declaration.
  - The number of studies making their datasets available has been growing over the past few years (Section 3.2.3.2).

  - The studies from which we extract claims (38 out of 101) have a notable overall quality (median score of 9 out of 11 — $\tilde{Qscore} = 9$).

6.1.2 Study 2: What is CI's empirically observable causal effect on software quality? (Chapter 4)

- We reviewed the literature cataloging relevant CI and software quality associations and adjacent relationships (Section 4.2.1).
- We built a literature-based causal DAG expressing the associations between CI and software quality (Section 4.2.1).
- We mined software repositories to empirically assess the literature-based causal DAG, analyzing releases in 12 activity months from 148 software projects (Section 4.2.2).
- Relying on the literature-based causal DAG (generated from the literature) and the statistical tests on the dataset, we proposed a new data-validated causal DAG (based on a hybrid literature-data approach) expressing the relationship between CI, software quality, and the relevant variables (Section 4.2.3).
- Findings and confirmations through the literature-data causal DAG:
  - CI has a positive causal effect on bug reports.
  - CI also influences developers' communication, reinforcing bug reports benefits.
  - The developers' overconfidence is a concern in CI environments and could negatively affect bug reports.
  - CI impacts on Merge Conflicts.
  - CI impacts on commit frequency.
  - Project age influences team communication.
  - Commit frequency, test volume, and communication are interrelated, and all of them affect Bug Report.

The studies are complementary. Each study complements a big picture of CI effects. Study 1 shows the literature-reported outcomes of CI on software developments in different themes. One of these outcomes is the software quality that Study 2 addressed under a causal perspective. Study 2 demonstrates that CI directly influences software quality and indirectly through different pathways — team communication, commit frequency, and test volume, which also influences bug reports. The overall conclusion is that CI causes an improvement in software quality (in terms of bug reports).

6.2 Future Work

Future studies can explore some areas identified in this study. This study focused on evaluating the effects of continuous integration concerning widely accepted software quality metrics. Future research may investigate other software development metrics to gain a more comprehensive view of CI impact on software quality.

In our empirical analysis, we examined open-source software projects. However, there is a diversity of software development projects, including commercial projects and a diversity of domains. Future research may investigate whether the effects of CI vary across different project types, providing additional insights into adopting the practice in various contexts. Additionally, this study focused on a specific CI platform, but numerous tools are available, such as GitHub Actions, Jenkins, CircleCI, and GitLab CI/CD. Future research can compare the effects of CI across different tools, identifying whether certain tools are more effective in specific contexts or for particular project types.

Furthermore, we point out several rooms for further investigation while presenting our systematic literature review in Chapter 3. Researchers can explore a more precise definition of CI and its practices to establish a solid foundation for further studies, helping to establish a more rigorous evaluation regarding CI adoption for researchers and to obtain more reliable results.

References

AMRIT, C.; MEIJBERG, Y. Effectiveness of test driven development and continuous integration: A case study. *IT Professional*, IEEE Computer Society, 6 2017. Cited 2 times on pages 114 and 116.

AMRIT, C.; MEIJBERG, Y. Effectiveness of test driven development and continuous integration x2013; a case study. *IT Professional*, IEEE Computer Society, 6 2017. ISSN 15209202. Cited 3 times on pages 78, 84, and 95.

BARNEY, S. et al. *Software quality trade-offs: A systematic map*. [S.l.]: Elsevier B.V., 2012. 651-662 p. Cited on page 25.

BECK, K.; ANDRES, C. *Extreme Programming Explained: Embrace Change (2nd Edition)*. [S.l.: s.n.], 2004. Cited 5 times on pages 17, 22, 25, 77, and 81.

BELLER, M.; GOUSIOS, G.; ZAIDMAN, A. Travistorrent: Synthesizing travis ci and github for full-stack research on continuous integration. In: . [S.l.]: IEEE Computer Society, 2017. p. 447–450. ISBN 9781538615447. ISSN 21601860. Cited on page 56.

BERNARDO, J. H.; COSTA, D. A. D.; KULESZA, U. Studying the impact of adopting continuous integration on the delivery time of pull requests. In: . [S.l.]: IEEE Computer Society, 2018. p. 131–141. ISBN 9781450357166. ISSN 02705257. Cited 2 times on pages 17 and 110.

BERNARDO, J. H. et al. The impact of a continuous integration service on the delivery time of merged pull requests. *Empirical Software Engineering*, v. 28, 2023. Cited on page 108.

BRINDESCU, C. et al. How do centralized and distributed version control systems impact software changes? In: *Proceedings of the 36th international conference on Software Engineering*. [S.l.: s.n.], 2014. p. 322–333. Cited on page 107.

CAIRO, A. S.; CARNEIRO, G. de F.; MONTEIRO, M. P. *The impact of code smells on software bugs: A systematic literature review*. [S.l.]: MDPI AG, 2018. Cited 2 times on pages 78 and 79.

CARTWRIGHT, N. *Nature's Capacities and their Measurement*. [S.l.]: Clarendon Press, 1989. Cited on page 93.

CASSEE, N.; VASILESCU, B.; SEREBRENIK, A. The silent helper: the impact of continuous integration on code reviews. In: . [S.l.: s.n.], 2020. p. 423–434. Cited 4 times on pages 74, 78, 95, and 108.

COHEN, J. *Psychological Bulletin WEIGHTED KAPPA: NOMINAL SCALE AGREEMENT WITH PROVISION FOR SCALED DISAGREEMENT OR PARTIAL CREDIT*. 1968. Cited 3 times on pages 34, 40, and 69.

CRUZES, D. S.; DYBå, T. Recommended steps for thematic synthesis in software engineering. In: . [S.l.]: IEEE Computer Society, 2011. p. 275–284. ISSN 19493789. Cited 3 times on pages 39, 40, and 71.

DEBBICHE, A.; DIENéR, M.; SVENSSON, R. B. Challenges when adopting continuous integration: A case study. In: . [S.l.]: Springer, Cham, 2014. p. 17–32. Cited on page 17.

DEBBICHE, A.; DIENéR, M.; SVENSSON, R. B. Challenges when adopting continuous integration: A case study. In: . [S.l.]: Springer, Cham, 2014. p. 17–32. Cited 2 times on pages 78 and 81.

DECAN, A. et al. On the use of github actions in software development repositories. In: IEEE. *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. [S.l.], 2022. p. 235–245. Cited on page 88.

DIKERT, K.; PAASIVAARA, M.; LASSENIUS, C. Challenges and success factors for large-scale agile transformations: A systematic literature review. *Journal of Systems and Software*, Elsevier Inc., v. 119, p. 87–108, 9 2016. ISSN 01641212. Cited 3 times on pages 112, 113, and 114.

DUVALL, P. M. *Continuous Delivery - Patterns and Anti-Patterns in the Software Lifecycle*. 2013. <https://dzone.com/refcardz/continuous-delivery-patterns>. Cited 10 times on pages 10, 17, 22, 23, 24, 30, 31, 41, 63, and 70.

DUVALL, P. M.; MATYAS, S.; GLOVER, A. *Continuous integration: improving software quality and reducing risk*. [S.l.]: Pearson Education, 2007. Cited 3 times on pages 17, 77, and 81.

DUVALL, P. M.; OLSON, M. Continuous delivery: Patterns and antipatterns in the software life cycle. *DZone refcard*, v. 145, 2011. Cited on page 64.

DYBå, T.; DINGSøYR, T.; HANSSEN, G. K. Applying systematic reviews to diverse study types: An experience report. In: . [S.l.: s.n.], 2007. p. 126–135. ISBN 0769528864. Cited on page 37.

DYBå, T.; TORGEIR, D. *Strength of Evidence in Systematic Reviews in Software Engineering*. [S.l.]: Association for Computing Machinery, 2008. 362 p. ISBN 9781595939715. Cited 2 times on pages 58 and 70.

EASTERBROOK, S. et al. *Selecting empirical methods for software engineering research*. [S.l.]: Springer, 2008. 285-311 p. Cited 2 times on pages 31 and 59.

EMBURY, S. M.; PAGE, C. Effect of continuous integration on build health in undergraduate team projects. In: . [S.l.]: Springer Verlag, 2019. v. 11350 LNCS, p. 169–183. ISBN 9783030060183. ISSN 16113349. Cited on page 17.

EYOLFSON, J.; TAN, L.; LAM, P. Correlations between bugginess and time-based commit characteristics. *Empirical Software Engineering*, Springer, v. 19, p. 1009–1039, 2014. Cited on page 108.

FELIDRé, W. et al. Continuous integration theater. *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 7 2019. Available at: <http://arxiv.org/abs/1907.01602>. Cited 3 times on pages 31, 64, and 91.

FERREIRA, M. A. et al. Faster issue resolution with higher technical quality of software. *Software Quality Journal*, Kluwer Academic Publishers, v. 20, p. 265–285, 2012. ISSN 15731367. Cited on page 79.

FITZGERALD, B.; STOL, K. J. Continuous software engineering and beyond: Trends and challenges. In: . [S.l.]: Association for Computing Machinery, Inc, 2014. p. 1–9. ISBN 9781450328562. Cited 2 times on pages 23 and 41.

FOWLER, M. Extreme programming. *Martinfowler. com. https://martinfowler. com/bliki/ExtremeProgramming.html*, 2013. Cited on page 129.

FOWLER, M. Continuousintegrationcertification. *Martinfowler. com. https://martinfowler. com/bliki/ContinuousIntegrationCertification.html. Accessed*, v. 26, 2020. Cited 4 times on pages 30, 31, 41, and 77.

FOWLER, M.; FOEMMEL, M. Continuous integration, 2006. *URL http://martinfowler. com/articles/continuousIntegration.html*, 2006. Cited 11 times on pages 10, 17, 22, 23, 24, 25, 31, 41, 63, 70, and 81.

GAROUSI, V.; MäNTYLä, M. V. *When and what to automate in software testing? A multi-vocal literature review*. [S.l.]: Elsevier B.V., 2016. 92-117 p. Cited 2 times on pages 81 and 82.

GARVIN, D. A. What does product quality really means? *Sloan management review*, v. 25, p. 25–43, 1984. Cited on page 25.

GHALEB, T. A.; COSTA, D. A. da; ZOU, Y. An empirical study of the long duration of continuous integration builds. *Empirical Software Engineering*, Springer New York LLC, v. 24, p. 2102–2139, 8 2019. ISSN 15737616. Cited on page 17.

GOLZADEH, M.; DECAN, A.; MENS, T. On the rise and fall of ci services in github. In: IEEE. *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. [S.l.], 2022. p. 662–672. Cited on page 88.

GREENLAND, S.; PEARL, J.; ROBINS, J. M. Causal diagrams for epidemiologic research. *Epidemiology*, JSTOR, p. 37–48, 1999. Cited on page 29.

GREENLAND, S.; PEARL, J.; ROBINS, J. M. Confounding and collapsibility in causal inference. *Statistical science*, Institute of Mathematical Statistics, v. 14, p. 29–46, 1999. Cited on page 74.

HEINZE-DEML, C.; PETERS, J.; MEINSHAUSEN, N. Invariant causal prediction for nonlinear models. *Journal of Causal Inference*, De Gruyter, v. 6, 2018. Cited 2 times on pages 92 and 97.

HERNáN, M. A.; ROBINS, J. M. *Causal inference*. [S.l.]: CRC Boca Raton, FL, 2010. Cited 8 times on pages 20, 26, 27, 28, 75, 86, 91, and 95.

HILTON, M.; BELL, J.; MARINOV, D. A large-scale study of test coverage evolution. In: . [S.l.: s.n.], 2018. p. 53–63. Cited 2 times on pages 82 and 95.

HILTON, M. et al. Trade-offs in continuous integration: Assurance, security, and flexibility. In: . [S.l.]: Association for Computing Machinery, 2017. Part F130154, p. 197–207. ISBN 9781450351058. Cited 3 times on pages 18, 114, and 116.

HILTON, M. et al. Usage, costs, and benefits of continuous integration in open-source projects. In: . [S.l.]: Association for Computing Machinery, Inc, 2016. p. 426–437. ISBN 9781450338455. Cited 6 times on pages 18, 22, 42, 87, 114, and 116.

HINDLE, A.; GERMAN, D. M.; HOLT, R. *What do large commits tell us?: a taxonomical study of large commits, in proceedings of the 2008 International Working Conference on Mining Software Repositories (MSR)*. [S.l.]: ACM, 2008. Cited on page 81.

HOLMSTROM, H. et al. Global software development challenges: A case study on temporal, geographical and socio-cultural distance. In: . [S.l.: s.n.], 2006. p. 3–11. Cited on page 17.

HOYER, R. W.; Y., B. B. Que es calidad clase 1 related papers. *Quality Progress*, v. 34, p. 53–62, 2001. Cited on page 25.

HUANG, A. F.; LIU, B. B.; HUANG, C. B. *A Taxonomy System to Identify Human Error Causes for Software Defects*. Cited 4 times on pages 78, 79, 80, and 95.

International Organization for Standardization. *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. Geneva, CH, 2011. v. 2011. Available at: <https://www.iso.org/standard/35733.html>. Cited on page 25.

ISLAM, M. R.; ZIBRAN, M. F. Insights into continuous integration build failures. In: . [S.l.]: IEEE Computer Society, 2017. p. 467–470. ISBN 9781538615447. ISSN 21601860. Cited on page 85.

JOHANSSEN, J. O. et al. Practitioners' eye on continuous software engineering: An interview study. In: . [S.l.]: Association for Computing Machinery, 2018. p. 41–50. ISBN 9781450364591. Cited on page 17.

KAUSAR, M.; AL-YASIRI, A. Distributed agile patterns for offshore software development. In: *12th International Joint Conference on Computer Science and Software Engineering (JCSSE), IEEE*. [S.l.: s.n.], 2015. Cited on page 17.

KAYNAK İlgi K.; ÇILDEN, E.; AYDIN, S. Software quality improvement practices in continuous integration. In: . [S.l.]: Springer Verlag, 2019. v. 1060, p. 507–517. ISBN 9783030280048. ISSN 18650937. Cited 5 times on pages 18, 74, 78, 114, and 116.

KEELE, S. et al. *Guidelines for performing systematic literature reviews in software engineering*. [S.l.], 2007. Cited 6 times on pages 19, 30, 35, 37, 69, and 70.

KERZAZI, N.; KHOMH, F.; ADAMS, B. Why do automated builds break? an empirical study. In: . [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2014. p. 41–50. ISBN 9780769553030. Cited 2 times on pages 78 and 85.

KHOMH, F. et al. Do faster releases improve software quality? an empirical case study of mozilla firefox. In: . [S.l.: s.n.], 2012. p. 179–188. ISBN 9781467317610. ISSN 21601852. Cited on page 26.

KITCHENHAM, B. A.; DYBå, T.; JøRGENSEN, M. Evidence-based software engineering. In: . [S.l.: s.n.], 2004. v. 26, p. 273–281. ISSN 02705257. Cited on page 31.

LAUKKANEN, E.; ITKONEN, J.; LASSENIUS, C. *Problems, causes and solutions when adopting continuous delivery—A systematic literature review*. [S.l.]: Elsevier B.V., 2017. 55-79 p. Cited 5 times on pages 19, 85, 112, 113, and 114.

LEPPäNEN, M. et al. The highways and country roads to continuous deployment. *IEEE Software*, v. 32, p. 64–72, 2015. Cited on page 31.

LERO, L. C. M. A. B. H. Z. *Towards an Evidence-Based Understanding of Electronic Data Sources*. 2010. Cited 2 times on pages 33 and 69.

LICORISH, S. A.; MACDONELL, S. G. Exploring software developers' work practices: Task differences, participation, engagement, and speed of task resolution. *Information and Management*, Elsevier B.V., v. 54, p. 364–382, 4 2017. ISSN 03787206. Cited 2 times on pages 80 and 81.

MARTIN, F.; MATTHEW, F. Continuous integration. *Recuperado de http://martinfowler. com/articles/continuousIntegration. html*, 2006. Cited on page 108.

MEEDENIYA, D. A.; RUBASINGHE, I. D.; PERERA, I. Software artefacts consistency management towards continuous integration: A roadmap. *International Journal of Advanced Computer Science and Applications*, Science and Information Organization, v. 10, p. 100–110, 2019. ISSN 21565570. Cited on page 17.

MOCKUS, A.; VOTTA, L. G. Identifying reasons for software changes using historic databases. In: . [S.l.: s.n.], 2000. p. 120–130. Cited 2 times on pages 80 and 81.

MUNAIAH, N. et al. Curating github for engineered software projects. *Empirical Software Engineering*, Springer, v. 22, p. 3219–3253, 2017. Cited on page 89.

MURGIA, A. et al. On the influence of maintenance activity types on the issue resolution time. In: . [S.l.]: Association for Computing Machinery, 2014. p. 12–21. ISBN 9781450328982. Cited on page 80.

NERY, G. S.; COSTA, D. A. da; KULESZA, U. An empirical study of the relationship between continuous integration and test code evolution. In: . [S.l.: s.n.], 2019. p. 426–436. Cited 7 times on pages 78, 82, 91, 92, 95, 107, and 110.

OLIVEIRA, M. C. de. Draco: Discovering refactorings that improve architecture using fine-grained co-change dependencies. In: . [S.l.: s.n.], 2017. p. 1018–1021. Cited on page 89.

OLIVEIRA, M. C. de et al. Finding needles in a haystack: Leveraging co-change dependencies to recommend refactorings. *Journal of Systems and Software*, Elsevier, v. 158, p. 110420, 2019. Cited on page 89.

PANJER, L. D. *Predicting Eclipse Bug Lifetimes*. 2007. Cited on page 80.

PARSONS, D.; RYU, H.; LAL, R. The impact of methods and techniques on outcomes from agile software development projects. In: SPRINGER. *IFIP International Working Conference on Organizational Dynamics of Technology-Based Innovation*. [S.l.], 2007. p. 235–249. Cited on page 18.

PEARL; JUDEA. *Title: Causal Diagrams for Empirical Research*. 1994. Available at: <http://escholarship.org/uc/item/6gv9n38c>. Cited 4 times on pages 20, 28, 75, and 86.

PEARL, J. et al. Models, reasoning and inference. *Cambridge, UK: CambridgeUniversityPress*, v. 19, 2000. Cited 9 times on pages 18, 26, 27, 28, 74, 75, 76, 86, and 95.

PEARL, J.; VERMA, T. S. A theory of inferred causation. In: *Studies in Logic and the Foundations of Mathematics*. [S.l.]: Elsevier, 1995. v. 134, p. 789–811. Cited 2 times on pages 27 and 86.

PEHMÖLLER, A.; SALGER, F.; WAGNER, S. Testing in global software development–a pattern approach. *arXiv preprint arXiv:2101.11317*, 2021. Cited on page 17.

PENROSE, O.; PERCIVAL, I. C. The direction of time. *Proceedings of the Physical Society (1958-1967)*, IOP Publishing, v. 79, p. 605, 1962. Cited 3 times on pages 26, 75, and 95.

PETERS, J.; JANZING, D.; SCHöLKOPF, B. *Elements of causal inference: foundations and learning algorithms*. [S.l.]: The MIT Press, 2017. Cited 2 times on pages 26 and 75.

PHALNIKAR, R.; DESHPANDE, V.; JOSHI, S. Applying agile principles for distributed software development. In: IEEE. *2009 International Conference on Advanced Computer Control*. [S.l.], 2009. p. 535–539. Cited on page 17.

PINTO, G. et al. Work practices and challenges in continuous integration: A survey with travis ci users. *Software - Practice and Experience*, John Wiley and Sons Ltd, v. 48, p. 2223–2236, 12 2018. ISSN 1097024X. Cited 6 times on pages 17, 18, 74, 78, 95, and 114.

PINTO, G.; REBOUÇAS, M.; CASTOR, F. Inadequate testing, time pressure, and (over) confidence: a tale of continuous integration users. In: IEEE. *2017 IEEE/ACM 10th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. [S.l.], 2017. p. 74–77. Cited on page 78.

RAFI, D. M. et al. Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. In: . [S.l.: s.n.], 2012. p. 36–42. ISBN 9781467318228. Cited on page 82.

RAHMAN, A. et al. Characterizing the influence of continuous integration: Empirical results from 250+ open source and proprietary projects. In: . [S.l.]: Association for Computing Machinery, Inc, 2018. p. 8–14. ISBN 9781450360562. Cited 4 times on pages 18, 74, 78, and 114.

RAUSCH, T. et al. An empirical analysis of build failures in the continuous integration workflows of java-based open-source software. *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 2017. Cited 2 times on pages 17 and 85.

REBOUCAS, M. et al. How does contributors' involvement influence the build status of an open-source software project? In: . [S.l.]: IEEE Computer Society, 2017. p. 475–478. ISBN 9781538615447. ISSN 21601860. Cited 2 times on pages 85 and 95.

ROBLES, G. Replicating msr: A study of the potential replicability of papers published in the mining software repositories proceedings. In: . [S.l.: s.n.], 2010. p. 171–180. Cited on page 57.

RODRIGUEZ-PéREZ, G.; ROBLES, G.; GONZáLEZ-BARAHONA, J. M. Reproducibility and credibility in empirical software engineering: A case study based on a systematic literature review of the use of the szz algorithm. *Information and Software Technology*, Elsevier, v. 99, p. 164–176, 2018. Cited 2 times on pages 31 and 57.

ROGERS, R. O. Scaling continuous integration. In: . [S.l.]: Springer Verlag, 2004. v. 3092, p. 68–76. ISBN 9783540221371. Cited on page 17.

RZIG, D. E. et al. Characterizing the usage of ci tools in ml projects. In: *Proceedings of the 16th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. [S.l.: s.n.], 2022. p. 69–79. Cited on page 88.

SANTOS, J.; COSTA, D. da; KULESZA, U. Investigating the impact of continuous integration practices on the productivity and quality of open-source projects. In: . [S.l.: s.n.], 2022. p. 137–147. Cited 4 times on pages 18, 77, 90, and 91.

SEO, H. et al. Programmers' build errors: A case study (at google). In: . [S.l.]: IEEE Computer Society, 2014. p. 724–734. ISSN 02705257. Cited on page 85.

SHAHIN, M.; BABAR, M. A.; ZHU, L. *Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices*. [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2017. 3909-3943 p. Cited 6 times on pages 19, 23, 41, 112, 113, and 114.

SHALIZI, C. R. Advanced data analysis from an elementary point of view. *URL http://www. stat. cmu. edu/˜ cshalizi/ADAfaEPoV*, 2021. Cited 3 times on pages 27, 28, and 86.

SHMUELI, G. To explain or to predict? *Statistical science*, Institute of Mathematical Statistics, v. 25, p. 289–310, 2010. Cited on page 29.

SOARES, E. et al. Slr artifacts - continuous integration quality impacts. 8 2021. Available at: <https://doi.org/10.5281/zenodo.4545623#.Y2Kvz3Tokno.mendeley>. Cited 2 times on pages 33 and 35.

SOARES, E. et al. The effects of continuous integration on software development: a systematic literature review. *Empirical Software Engineering*, Springer, v. 27, n. 3, p. 1–61, 2022. Cited 7 times on pages 17, 20, 30, 74, 78, 81, and 91.

SOMMERVILLE, I. et al. Engenharia de software.[sl]. *Pearson Education*, v. 19, p. 23, 2011. Cited on page 25.

SPIRTES, P.; GLYMOUR, C.; SCHEINES, R. *Causation, Prediction, and Search*. Springer New York, 1993. ISBN 978-1-4612-7650-0. Available at: <http://link.springer.com/10.1007/978-1-4612-2748-9>. Cited 3 times on pages 26, 27, and 93.

STAHL, D.; BOSCH, J. Automated software integration flows in industry: A multiple-case study. In: . [S.l.]: Association for Computing Machinery, 2014. p. 54–63. ISBN 9781450327688. Cited on page 17.

STåHL, D.; BOSCH, J. Experienced benefits of continuous integration in industry software product development: A case study. In: . [S.l.: s.n.], 2013. p. 736–743. ISBN 9780889869431. Cited 7 times on pages 17, 18, 19, 112, 113, 114, and 116.

STåHL, D.; BOSCH, J. Modeling continuous integration practice differences in industry software development. *Journal of Systems and Software*, v. 87, p. 48–59, 1 2014. ISSN 01641212. Cited 2 times on pages 17 and 74.

STåHL, D.; BOSCH, J. Modeling continuous integration practice differences in industry software development. *Journal of Systems and Software*, v. 87, p. 48–59, 1 2014. ISSN 01641212. Cited 12 times on pages 23, 30, 31, 41, 57, 64, 69, 70, 73, 112, 113, and 114.

TEXTOR, J. et al. Robust causal inference using directed acyclic graphs: the r package 'dagitty'. *International journal of epidemiology*, Oxford University Press, v. 45, p. 1887–1894, 2016. Cited on page 86.

THOUGHTWORKS. *CI theatre*. 2017. Available at: <https://www.thoughtworks.com/radar/techniques/ci-theatre>. Cited 2 times on pages 64 and 91.

VASILESCU, B. et al. Quality and productivity outcomes relating to continuous integration in github. In: . [S.l.]: Association for Computing Machinery, Inc, 2015. p. 805–816. ISBN 9781450336758. Cited 17 times on pages 17, 18, 20, 22, 26, 63, 74, 75, 77, 78, 79, 84, 90, 95, 96, 114, and 116.

VASSALLO, C.; PALOMBA, F.; GALL, H. C. Continuous refactoring in ci: A preliminary study on the perceived advantages and barriers. In: . [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2018. p. 564–568. ISBN 9781538678701. Cited 2 times on pages 17 and 23.

VASSALLO, C. et al. Automated reporting of anti-patterns and decay in continuous integration. In: . [S.l.]: IEEE Computer Society, 2019. v. 2019-May, p. 105–115. ISBN 9781728108698. ISSN 02705257. Cited 2 times on pages 17 and 64.

VASSALLO, C. et al. Every build you break: developer-oriented assistance for build failure resolution. *Empirical Software Engineering*, Springer, v. 25, p. 2218–2257, 5 2020. ISSN 15737616. Cited on page 85.

VIGGIATO, M. et al. Understanding similarities and differences in software development practices across domains. In: . [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2019. p. 84–94. ISBN 9781538691960. Cited 8 times on pages 23, 30, 31, 41, 57, 69, 70, and 73.

VOLF, Z.; SHMUELI, E. Screening heuristics for project gating systems. In: . [S.l.]: Association for Computing Machinery, 2017. Part F130154, p. 872–877. ISBN 9781450351058. Cited on page 17.

WHEELAN, S. A.; HOCHBERGER, J. M. Validation studies of the group development questionnaire. *Small group research*, SAGE PUBLICATIONS, INC. 2455 Teller Road, Thousand Oaks, CA 91320, v. 27, p. 143–170, 1996. Cited on page 68.

WIDDER, D. G. et al. I'm leaving you, travis: a continuous integration breakup story. In: *Proceedings of the 15th International Conference on Mining Software Repositories*. [S.l.: s.n.], 2018. p. 165–169. Cited on page 88.

YU, Y. et al. Determinants of pull-based development in the context of continuous integration. *Science China Information Sciences*, Science in China Press, v. 59, 8 2016. ISSN 1674733X. Cited on page 17.

ZAIDMAN, A. et al. Mining software repositories to study co-evolution of production test code. In: . [S.l.: s.n.], 2008. p. 220–229. Cited 2 times on pages 82 and 95.

ZAIDMAN, A. et al. Studying the co-evolution of production and test code in open source and industrial developer test processes through repository mining. *Empirical Software Engineering*, Springer, v. 16, p. 325–364, 2011. Cited 2 times on pages 82 and 95.

ZAMPETTI, F. et al. How open source projects use static code analysis tools in continuous integration pipelines. In: . [S.l.]: IEEE Computer Society, 2017. p. 334–344. ISBN 9781538615447. ISSN 21601860. Cited on page 17.

ZAYTSEV, Y. V.; MORRISON, A. Increasing quality and managing complexity in neuroinformatics software development with continuous integration. *Frontiers in Neuroinformatics*, v. 6, 1 2013. ISSN 16625196. Cited on page 114.

ZHANG, F. et al. An empirical study on factors impacting bug fixing time. In: . [S.l.: s.n.], 2012. p. 225–234. ISBN 9780769548913. ISSN 10951350. Cited on page 80.

ZHANG, T. et al. *A Literature Review of Research in Bug Resolution: Tasks, Challenges and Future Directions*. [S.l.]: Oxford University Press, 2016. 741-773 p. Cited 2 times on pages 79 and 80.

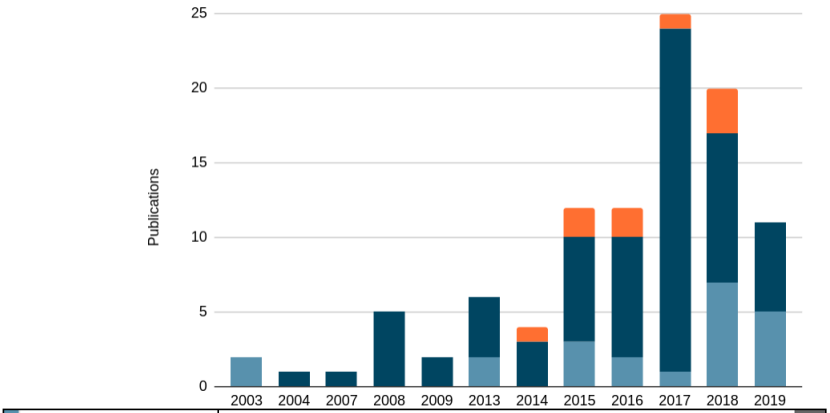
ZHAO, Y. et al. The impact of continuous integration on other software development practices: A large-scale empirical study. In: . [S.l.]: IEEE / ACM, 2017. ISBN 9781538626849. Cited 10 times on pages 22, 30, 31, 41, 69, 70, 74, 78, 91, and 107.

APPENDIX A – Systematic Literature Review on The Effects of Continuous Integration on Software Development

A.1 Demographic attributes

This appendix shows the demographic data of our primary studies. We discuss the evolution of studies over the years and describe the authors' information next.

Evolution of studies. CI emerged in the context of eXtreme Programming, a software development methodology that increased and became popular in the late 90s and early 00s (FOWLER, 2013). Indeed, we identify the first research efforts on CI in 2003. Figure 33 shows an increasing number of publications over the years, especially in the last five years. The majority of papers have been published in conference proceedings (69 papers, i.e., 67.2%), followed by 23 papers published in journals (i.e. 22.5%). 10 other studies have been published in workshops in the last years (i.e. 9.8%).



	2003	2004	2007	2008	2009	2013	2014	2015	2016	2017	2018	2019	
Journal Article	2	0	0	0	0	2	0	3	2	1	7	5	22
Conference Proceedings	0	1	1	5	2	4	3	7	8	23	10	6	70
Workshop	0	0	0	0	0	0	1	2	2	1	3	0	9

Figure 33 – Publications by year and type of venue.

Our primary studies have been published in 29 distinct conferences, 15 journals, and 7 workshops. Figure 34 (a) shows that MSR (IEEE International Working Conference on Mining Software Repositories), ICSE (International Conference on Software Engineering), Agile Conference, and ESEC/FSE (European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering) are the conferences

with the highest number of primary studies. Since we aim to collate the most claims possible related to CI, we do not necessarily focus on the goals of a venue (e.g., magazine-based publication). In a later stage, we analyse the rigour of the studies from which we find claims.

As for workshops, Figure 34 (b) shows Conference XP (Scientific Workshops Proceedings), SWAN (International Workshop on Software Analytics), and RCoSE (International Workshop on Rapid Continuous Software Engineering) as the most frequent venues. Figure 34 (c) shows that the journals with the highest frequency are Empirical Software Engineering, Information and Software Technology, and IEEE Software.

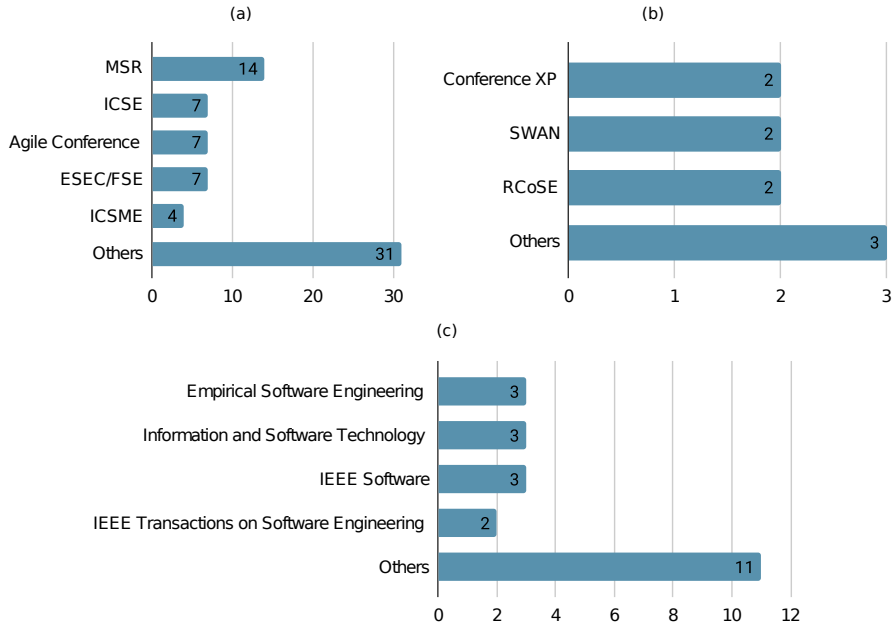


Figure 34 – Publications in main venues on (a) conferences, (b) Workshops, and (c) Journals.

Paper Authors. The primary studies have 259 different authors involved altogether. Table 38 shows a ranking with those having the highest number of publications included as a primary study in our SLR. Jan Bosch is the most frequent author and all of the top 6 researchers remain active over the last years. Having described the demographic data of our primary studies, we now describe our obtained results.

A.2 Selected Studies

ID	Title	Author(s)	Year	Venue
P2	(No) influence of continuous integration on the commit activity in GitHub projects	Stephan Diehl, Daniel Anastasiou, Jascha Knack, Sebastian Baltes, Ralf Tymann	2018	SWAN - International Workshop on Software Analytics
P3	A brief study on build failures in continuous integration: Causation and effect	Bharavi Mishra, Saket Kumar Singh, Romit Jain	2018	ICACIE - Progress in Advanced Computing and Intelligent Engineering

P4	A conceptual replication of continuous integration pain points in the context of Travis CI	Bogdan Vasilescu, David Gray Widder, Michael Hilton, Christian Kästner	2019	ESEC/FSE Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering
P5	A Current Study on the Limitations of Agile Methods in Industry Using Secure Google Forms	Ashish Agrawal, L. S. Maurya, Mohd Aurangzeb Atiq	2016	International Conference on Information Security and Privacy
P6	A Hundred Days of Continuous Integration	Ade MillerAde Miller	2008	Agile Conference
P7	A Study on the Interplay between Pull Request Review and Continuous Integration Builds	Massimiliano Di Penta, Canfora Gerardo, Gabriele Bavota, Fiorella Zampetti	2019	SANER
P8	A Tale of CI Build Failures: An Open Source and a Financial Organization Perspective	Carmine Vassallo, Gerald Schermann, Fiorella Zampetti, Daniele Romano, Philipp Leitner, Andy Zaidman, Massimiliano Di Penta, Sebastiano Panichella	2017	ICSME - International Conference on Software Maintenance and Evolution
P9	Agile systems development and stakeholder satisfaction: a South African empirical study	Jason Cohen, Carlos Ferreira	2008	SAICSIT
P10	An empirical analysis of build failures in the continuous integration workflows of Java-based open-source software	Stefan Schulte, Thomas Rausch, Waldemar Hummer, Philipp Leitner	2017	MSR - International Conference on Mining Software Repositories
P11	An empirical study of activity, popularity, size, testing, and stability in continuous integration	Saket Vishwasrao, Francisco Servant, Aakash Gautam	2017	MSR - International Conference on Mining Software Repositories
P12	An empirical study of the long duration of continuous integration builds	Ying Zou, Daniel Alencar da Costa, Taher Ahmed Ghaleb	2019	Empirical Software Engineering
P13	An empirical study of the personnel overhead of continuous integration	Shane McIntosh, Eduardo Coronado-Montoya, Marco Manglaviti, Keheliya Gallaba	2017	MSR - International Conference on Mining Software Repositories
P14	Analyzing the effects of test driven development in GitHub	Abram Hindle, Neil Borle, Meysam Feghhi, Eleni Stroulia, Russ Greiner	2018	Empirical Software Engineering
P15	Analyzing the impact of social attributes on commit integration success	Mauricio Soto, Zack Coker, Claire Le Goues	2017	MSR - International Conference on Mining Software Repositories
P16	Angry-builds: an empirical study of affect metrics and builds success on github ecosystem	Michele Marchesi, David Bowes, Giuseppe Destefanis, Marco Ortu, Andrea Pinna, Roberto Tonelli	2018	Conference XP
P17	Applying Continuous Integration for Reducing Web Applications Development Risks	Fang Yie Leu, Sen Tarng Lai	2015	BWCCA - International Conference on Broadband and Wireless Computing, Communication and Applications
P18	Automated reporting of anti-patterns and decay in continuous integration	Sebastian Proksch, Harald C. Gall, Massimiliano Di Penta, Carmine Vassallo	2019	ICSE - International Conference on Software Engineering
P19	Automated software integration flows in industry: a multiple-case study	Daniel Ståhl, Jan Bosch	2014	ICSE - International Conference on Software Engineering

P20	Build waiting time in continuous integration: an initial interdisciplinary literature review	Mika Mantyla, Eero Laukkanen	2015	RCoSE - International Workshop on Rapid Continuous Software Engineering
P21	Building lean continuous integration and delivery pipelines by applying DevOps principles: a case study at Varidesk	Vidroha Debroy, Senecca Miller, Lance Brimble	2018	ESEC/FSE Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering
P22	Building lean thinking in a telecom software development organization: strengths and challenges	Pasi Kuvaja, Pilar Rodríguez, Kirsi Mikkonen, Markku Oivo, Juan Garbajosa	2013	ICSSP - International Conference on Software and System Process
P23	Built to last or built too fast? evaluating prediction models for build times	Ekaba Bisong, Eric Tran, Olga Baysal	2017	MSR - International Conference on Mining Software Repositories
P24	Challenges When Adopting Continuous Integration: A Case Study	Mikael Dienér, Richard Berntsson Svensson, Adam Debbiche	2014	International Conference on Product-Focused Software Process Improvement
P25	Characterizing the influence of continuous integration: empirical results from 250+ open source and proprietary projects	Akond Rahman, Amritanshu Agrawal, Rahul Krishna, Alexander Sobran	2018	SWAN - International Workshop on Software Analytics
P27	Comparison of release engineering practices in a large mature company and a startup	Eero Laukkanen, Casper Lassenius, Juha Itkonen, Maria Paasivaara	2018	Empirical Software Engineering
P28	Continuous code quality: are we (really) doing that?	Alberto Bacchelli, Harald C. Gall, Fabio Palomba, Carmine Vassallo	2018	ASE - International Conference on Automated Software Engineering
P29	Continuous Delivery Practices in a Large Financial Organization	Andy Zaidman, Carmine Vassallo, Fiorella Zampetti, Daniele Romano, Moritz Beller, Annibale Panichella, Massimiliano Di Penta	2017	ICSME - International Conference on Software Maintenance and Evolution
P30	Continuous Delivery: Huge Benefits, but Challenges Too	Lianping Chen	2015	IEEE Software
P31	Continuous Delivery? Easy! Just Change Everything (Well, Maybe It Is Not That Easy)	Steve Neely, Steve Stolt	2013	Agile Conference
P32	Continuous deployment and schema evolution in SQL databases	Michael De Jong, Arie Van Deursen	2015	RELENG - International Workshop on Release Engineering
P33	Continuous deployment at Facebook and OANDA	Michael Gentili, Kent Beck, Laurie Williams, Michael Stumm, Tony Savor, Mitchell Douglas	2016	ICSE - International Conference on Software Engineering
P34	Continuous deployment of mobile software at facebook (showcase)	Elisa Shibley, Chuck Rossi, Kent Beck, Shi Su, Michael Stumm, Tony Savor	2016	ESEC/FSE Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering
P36	Continuous Integration and Delivery for HPC: Using Singularity and Jenkins	Zebula Sampedro, Aaron Holt, Thomas Hauser	2018	PEARC - Practice and Experience on Advanced Research Computing
P37	Continuous Integration and Quality Assurance: a case study of two open source projects	Jesper Holck, Niels Jørgensen	2003	Australasian Journal of Information Systems

P38	Continuous Integration Applied to Software-Intensive Embedded Systems – Problems and Experiences	Torvald Mårtensson, Daniel Ståhl, Jan Bosch	2016	International Conference on Product-Focused Software Process Improvement
P39	Continuous Integration for Web-Based Software Infrastructures: Lessons Learned on the webinos Project	John Lyle, Tao Su, Andrea Atzeni, Shamal Faily, Habib Virji, Christos Ntanos, Christos Botsikas	2013	Haifa Verification Conference
P40	Continuous Integration Impediments in Large-Scale Industry Projects	Torvald Mårtensson, Jan Bosch, Daniel Ståhl	2017	ICSA - IEEE International Conference on Software Architecture
P41	Continuous Integration in a Social-Coding World: Empirical Evidence from GitHub	Bogdan Vasilescu, Mark G J Van Den Brand, Jules Wulms, Stef Van Schuylenburg, Alexander Serebrenik	2014	ICSME - International Conference on Software Maintenance and Evolution
P42	Continuous Integration in Open Source Software Development	Amit Deshpande, Dirk Riehle	2008	IFIP International Federation for Information Processing
P43	Continuous Integration is Not About Build Systems	Torvald Mårtensson, Par Hammarstrom, Jan Bosch	2017	SEAA - Euromicro Conference on Software Engineering and Advanced Applications
P44	Continuous Refactoring in CI: A Preliminary Study on the Perceived Advantages and Barriers	Carmine Vassallo, Fabio Palomba, Harald C. Gall	2018	ICSME - International Conference on Software Maintenance and Evolution
P45	Continuous software engineering and beyond: trends and challenges	Brian Fitzgerald, Klaas Jan Stol	2014	RCoSE - International Workshop on Rapid Continuous Software Engineering
P46	Contrasting Big Bang with Continuous Integration Through Defect Reports	Daniel Levin, Ana Magazinius, Niklas Mellegard, Hakan Burden, Kenneth Lind	2018	IEEE Software
P47	Determinants of pull-based development in the context of continuous integration	Cheng Yang, Huaimin Wang, Tao Wang, Gang Yin, Yue Yu	2016	Science China Information Sciences
P48	DevOps: A Definition and Perceived Adoption Impediments	Kristian Nybom, Jens Smeds, Ivan Porres	2015	International Conference on Agile Software Development
P49	Effect of Continuous Integration on Build Health in Undergraduate Team Projects	Suzanne M. Embury, Christopher Page	2017	Conference on Software Engineering Education and Training
P50	Effectiveness of Test-Driven Development and Continuous Integration: A Case Study	Yoni Meijberg, Chintan Amrit	2018	IT Professional
P51	Enabling Agile Testing through Continuous Integration	Sean Stolberg	2009	Agile Conference
P52	Experienced benefits of continuous integration in industry software product development: A case study	Jan Bosch, Daniel Ståhl	2013	IASTED International Conference on Software Engineering
P53	How does contributors' involvement influence the build status of an open-source software project?	Renato O. Santos, Fernando Castor, Gustavo Pinto, Marcel Reboucas	2017	MSR - International Conference on Mining Software Repositories
P54	How open source projects use static code analysis tools in continuous integration pipelines	Fiorella Zampetti, Rocco Oliveto, Gerardo Canfora, Massimiliano Di Penta, Simone Scalabrino	2017	MSR - International Conference on Mining Software Repositories

P55	I'm leaving you, Travis: a continuous integration breakup story	Bogdan Vasilescu, Christian Kästner, Michael Hilton, David Gray Widder	2018	ICSE - International Conference on Software Engineering
P56	Impact of continuous integration on code reviews	Mohammad Masudur Rahman, Chanchal K. Roy	2017	MSR - International Conference on Mining Software Repositories
P57	Implementation of a DevOps Pipeline for Serverless Applications	Vitalii Ivanov, Kari Smolander	2018	International Conference on Product-Focused Software Process Improvement
P58	Inadequate testing, time pressure, and (over) confidence: a tale of continuous integration users	Marcel Reboucas, Gustavo Pinto, Fernando Castor	2017	CHASE - International Workshop on Cooperative and Human Aspects of Software Engineering
P59	Increasing quality and managing complexity in neuroinformatics software development with continuous integration	Yury V. Zaytsev, Abigail Morrison	2013	Frontiers in Neuroinformatics
P60	Industry application of continuous integration modeling: a multiple-case study	Daniel Ståhl, Jan Bosch	2016	ICSE - International Conference on Software Engineering
P62	Insights into continuous integration build failures	Md Rakibul Islam, Minhaz F. Zibran	2017	MSR - International Conference on Mining Software Repositories
P63	ISM based identification of quality attributes for agile development	Parita Jain, Laxmi Ahuja, Arun Sharma	2016	International Conference on Reliability
P64	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	David Goodman, Michael Elbaz	2008	Agile Conference
P65	Lessons Learned: Using a Static Analysis Tool within a Continuous Integration System		2016	ISSREW - International Symposium on Software Reliability Engineering Workshops
P66	Managing to release early, often and on time in the OpenStack software ecosystem	José Apolinário Teixeira, Helena Karsten	2019	Journal of Internet Services and Applications
P67	Measurement and Impact Factors of Speed of Reviews and Integration in Continuous Software Engineering	Wilhelm Meding, Ola Söder, Magnus Bäck, Miroslaw Staron	2018	Foundations of Computing and Decision Sciences
P69	Moving from Closed to Open Source: Observations from Six Transitioned Projects to GitHub	Pavneet Singh Kochhar, Nachiappan Nagappan, Eirini Kalliamvakou, Christian Bird, Thomas Zimmermann	2019	IEEE Transactions on Software Engineering
P70	On the interplay between non-functional requirements and builds on continuous integration	Marcelo De A. Maia, Cricia Z. Felicio, Klerisson V.R. Paixao, Fernanda M. Delfim	2017	MSR - International Conference on Mining Software Repositories
P71	On the journey to continuous deployment: Technical and social challenges along the way	Gerry Gerard Claps, Richard Berntsson Svensson, Aybüke Aurum	2015	Information and Software Technology
P72	Oops, my tests broke the build: an explorative analysis of Travis CI with GitHub	Moritz Beller, Andy Zaidman, Georgios Gousios	2017	MSR - International Conference on Mining Software Repositories

P73	Practitioners' eye on continuous software engineering: An interview study	Jan Ole Johanssen, Anja Kleebaum, Bernd Bruegge, Barbara Paech	2018	ICSSP - International Conference on Software and System Process
P74	Quality and productivity outcomes relating to continuous integration in GitHub	Vladimir Filkov, Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu	2015	ESEC/FSE Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering
P75	Scaling Continuous Integration	R. Owen Rogers	2004	International Conference on Extreme Programming and Agile Processes in Software Engineering
P76	Screening heuristics for project gating systems	Edi Shmueli, Zahy Volf	2017	ESEC/FSE Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering
P77	Sentiment analysis of Travis CI builds	Bruno Silva, Rodrigo Souza	2017	MSR - International Conference on Mining Software Repositories
P78	Software artefacts consistency management towards continuous integration: A roadmap	I. Perera, D. A. Meedeniya, I. D. Rubasinghe	2019	International Journal of Advanced Computer Science and Applications
P79	Software Quality Improvement Practices in Continuous Integration	Selin Aydin, İlgi Keskin Kaynak, Evren Çilden	2019	European Conference on Software Process Improvement
P80	Stakeholder Perceptions of the Adoption of Continuous Integration – A Case Study	Maria Paasivaara, Teemu Arvonen, Eero Laukkanen	2015	Agile Conference
P81	Studying the impact of adopting continuous integration on the delivery time of pull requests	Joao Helis Bernardo, Uirá Kulesza, Daniel Alencar da Costa	2018	ICSE - International Conference on Software Engineering
P82	Successful extreme programming: Fidelity to the methodology or good teamworking?	Stephen Wood, George Michaelides, Chris Thomson	2013	Information and Software Technology
P83	Synthesizing Continuous Deployment Practices Used in Software Development	Chris Parnin, Akond Rahman, Eric Helms, Laurie Williams	2015	Agile Conference
P84	Team Pace Keeping Build Times Down	Graham Brooks	2008	Agile Conference
P85	Test activities in the continuous integration and delivery pipeline	Daniel Ståhl, Torvald Mårtensson, Jan Bosch	2019	Journal of Software: Evolution and Process
P86	The continuity of continuous integration: Correlations and consequences	Jan Bosch, Torvald Mårtensson, Daniel Ståhl	2017	Journal of Systems and Software
P87	The effects of individual XP practices on software development effort	Paul Rodrigues, Prakash Ramaswamy, S Kuppuswami, K Vivekanandan	2003	ACM SIGSOFT Software Engineering Notes
P88	The highways and country roads to continuous deployment	Marko Leppänen, Mika V Mäntylä, Juha Itkonen, Veli-Pekka Eloranta, Max Pagels, Simo Mäkinen, Tomi Männistö	2015	IEEE Software

P89	The impact of continuous integration on other software development practices: a large-scale empirical study	Vladimir Filkov, Yuming Zhou, Alexander Serebrenik, Yangyang Zhao, Bogdan Vasilescu	2017	ASE - International Conference on Automated Software Engineering
P90	The impact of the adoption of continuous integration on developer attraction and retention	Keheliya Gallaba, Yash Gupta, Yusaira Khan, Shane McIntosh	2017	MSR - International Conference on Mining Software Repositories
P91	The links between agile practices, interpersonal conflict, and perceived productivity	Lucas Gren	2017	EASE - Conference on Evaluation and Assessment in Software Engineering
P92	An empirical study examining the usage and perceived importance of XP practices	Jessica Zhang, Ann Fruhling	2007	AMCIS - Americas Conference on Information Systems
P93	The Tarpit – A general theory of software engineering	Pontus Johnson, Mathias Ekstedt	2016	Information and Software Technology
P94	Towards Agile Testing for Railway Safety-critical Software	Jin Guo, Yaxin Cao, Chang Rao, Yao Li, Nan Li, Jeff Lei	2016	Conference XP
P95	Towards Architecting for Continuous Delivery	Lianping Chen	2015	ICSA - IEEE International Conference on Software Architecture
P96	Towards quality gates in continuous delivery and deployment	Gerald Schermann, Jürgen Cito, Harald C. Gall, Philipp Leitner	2016	ICPC
P97	Trade-offs in continuous integration: assurance, security, and flexibility	Danny Dig, Michael Hilton, Nicholas Nelson, Timothy Tunnell, Darko Marinov	2017	ESEC/FSE Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering
P98	Transparency and contracts: continuous integration and delivery in the automotive ecosystem	Eric Knauss, Rob Van Der Valk, Patrizio Pelliccione, Rogardt Heldal, Patricia Lago, Jacob Juul	2018	ICSE - International Conference on Software Engineering
P99	Understanding similarities and differences in software development practices across domains	Pooyan Jamshidi, Christian Kästner, Markos Viggiato, Eduardo Figueiredo, Johnatan Oliveira	2019	ICGSE - International Conference on Global Software Engineering
P100	Usage, costs, and benefits of continuous integration in open-source projects	Timothy Tunnell, Michael Hilton, Kai Huang, Darko Marinov, Danny Dig	2016	ASE - International Conference on Automated Software Engineering
P101	Use and Misuse of Continuous Integration Features: An Empirical Study of Projects that (mis)use Travis CI	Keheliya Gallaba, Shane McIntosh	2018	IEEE Transactions on Software Engineering
P102	Using continuous integration and automated test techniques for a robust C4ISR system	Eray Tüzün, Erdoğan Gelirli, H. Mehmet Yüksel, Emrah Biyikli, Buyurman Baykal	2009	ISCIS - International Symposium on Computer and Information Sciences
P103	Vulnerabilities in Continuous Delivery Pipelines? A Case Study	Christina Paule, Thomas F. Dullmann, Andre Van Hoorn	2019	ICSA - IEEE International Conference on Software Architecture
P104	Wait for it: determinants of pull request evaluation latency on GitHub	Yue Yu, Bogdan Vasilescu, Premkumar Devanbu, Vladimir Filkov, Huaimin Wang	2015	MSR - International Conference on Mining Software Repositories

P105	Why modern open source projects fail	Jailton Coelho, Marco Tulio Valente	2017	ESEC/FSE Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering
P106	Work practices and challenges in continuous integration: A survey with Travis CI users	Rodrigo Bonifacio, Marcel Reboucas, Gustavo Pinto, Fernando Castor	2018	Software - Practice and Experience

Table 39 – Primary Studies selected in the review.

Table 38 – Ranking of authors per publication number and his publications.

	Researcher	**#**	**Publications**
1	Jan Bosch	8	2013 (P52), 2014 (P19), 2016 (P60, P38), 2017 (P86, P43, P40), 2019 (P85)
2	Daniel Ståhl	7	2013 (P52), 2014 (P19), 2016 (P38, P60), 2017 (P86, P40), 2019 (P85)
3	Bogdan Vasilescu	6	2014 (P41), 2015 (P104, P74), 2017 (P89), 2018 (P55), 2019 (P4)
4	Massimiliano Di Penta	5	2017 (P54, P8, P29), 2019 (P18, P7)
	Carmine Vassallo	5	2017 (P29, P8), 2018 (P44, P28), 2019 (P18)
	Torvald Mårtensson	5	2016 (P38), 2017 (P43, P40, P86), 2019 (P85)