

# Tacit Inefficiencies and Barriers in Continuous Integration

by

Nimmi Rashinika Weeraddana

A thesis  
presented to the University of Waterloo  
in fulfillment of the  
thesis requirement for the degree of  
Doctor of Philosophy  
in  
Computer Science

Waterloo, Ontario, Canada, 2025

© Nimmi Rashinika Weeraddana 2025

## Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner: Benoit Baudry  
Professor,  
Département d'informatique et de recherche opérationnelle,  
Université de Montréal

Supervisor: Shane McIntosh  
Associate Professor,  
Department of Computer Science,  
University of Waterloo

Internal Members: Michael W. Godfrey  
Professor,  
Department of Computer Science,  
University of Waterloo

Chengnian Sun  
Associate Professor,  
Department of Computer Science,  
University of Waterloo

Internal-External Member: Ladan Tahvildari  
Professor,  
Department of Electrical and Computer Engineering,  
University of Waterloo

## **Author's Declaration**

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

## Statement of Contributions

Earlier versions of the work in this thesis have been published as follows:

- **Weeraddana NR**, Alfadel M, McIntosh S. Characterizing Timeout Builds in Continuous Integration. *Transactions on Software Engineering*, vol 50(6), 2024 [174].
- **Weeraddana NR**, Alfadel M, McIntosh S. Dependency-Induced Waste in Continuous Integration: An Empirical Study of Unused Dependencies in the npm Ecosystem. *International Symposium on the Foundations of Software Engineering*, 2024 [175].
- **Weeraddana NR**, Xu X, Alfadel M, McIntosh S, Nagappan M. An Empirical Comparison of Ethnic and Gender Diversity of DevOps and non-DevOps Contributions to Open-Source Projects. *Empirical Software Engineering*, vol 28(6), 2023 [177].

I am the primary author for the above publications as well as the research presented in this thesis. My responsibilities include: (1) formulating the research ideas, (2) collecting data, (3) conducting empirical studies, and (4) drafting the manuscripts. My co-authors contributed by helping to refine the research ideas, manual coding, and providing constructive feedback to improve the manuscripts.

The following publications, while not directly related to the material in this thesis, were produced concurrently with the research conducted for this thesis.

- **Weeraddana NR**, Habchi S, McIntosh S. Crash Report Prioritization for Large-Scale Scheduled Launches. *International Conference on Software Engineering: Software Engineering in Practice*, 2025 [176].
- Kola-Olawuyi A, **Weeraddana NR**, Nagappan M. The Impact of Code Ownership of DevOps Artefacts on the Outcome of DevOps CI Builds. *International Conference on Mining Software Repositories*, 2024 [94].

## Abstract

Continuous Integration (CI) is the heartbeat of a software project. CI enables team members to validate each change set through an automated cycle (i.e., a CI build) that compiles and tests the project’s source code. Although adoption of CI improves team productivity and software quality, these benefits come at a cost. As projects evolve, the complexity of CI pipelines tends to increase, introducing potential inefficiencies (i.e., prolonged build durations and frequent build restarts) and barriers (e.g., the specialized expertise required to maintain CI artifacts). Such inefficiencies and barriers waste resources that enable CI.

While inefficiencies and barriers in CI are often explicit, where project teams are cognizant of them, there also exist tacitly accrued inefficiencies and barriers that are not immediately apparent to project teams. In this thesis, we use historical data from a large collection of software projects to perform three empirical studies, focusing on tacit inefficiencies and barriers in CI.

We first present an empirical study that focuses on tacit inefficiencies in the environment (e.g., CircleCI) where CI builds are executed. We observe that (1) CI builds can unexpectedly time out due to issues in the environment, such as network problems and resource constraints, and (2) the history of previous CI build outcomes and anticipation of clusters of consecutive timeouts can provide useful indications to project teams to proactively allocate resources and take preventive measures. Next, we present an empirical study that investigates tacit inefficiencies in CI that stem from dependencies in projects (e.g., npm dependencies). More specifically, CI builds triggered from change sets that update versions of unused dependencies are entirely wasteful because such change sets do not impact the project source code. We find that (1) a substantial amount of CI build time is spent on these wasteful builds, (2) bots that automatically manage dependency updates in projects (e.g., Dependabot) need to consider whether a dependency is used before triggering a build, and (3) to detect and omit such wasteful builds, project teams may adopt our automated approach, DEP-SCIMITAR<sup>1</sup>, to cut down on this waste.

We then present an empirical study that investigates tacit barriers that are related to the composition of the teams responsible for creating and maintaining CI pipelines, i.e., the DevOps contributors. In particular, we examine the diversity and inclusion of these contributors—a factor that plays a crucial role in CI by influencing collaboration and the overall efficiency of CI pipelines. Our findings show that (1) the perceived ethnic diversity of DevOps contributors is significantly low compared to other contributors, with a similar pattern observed for perceived gender diversity, and (2) the lack of diversity is amplified when considering the intersection of minority ethnicities and genders, calling for enhanced awareness of the lack of diversity among DevOps contributors.

## Acknowledgements

I am incredibly thankful for all the wonderful people who supported and inspired me throughout my PhD.

First and foremost, my deepest thanks go to my supervisor, Shane McIntosh. I have learned so much under your supervision, and I am truly grateful for your mentoring, support, and courage.

I would like to extend my sincere appreciation to Mahmoud Alfadel, Meiyappan Nagappan, and Sarra Habchi for their thoughtful feedback, ideas, and collaboration.

To my defense committee, Benoit Baudry, Chengnian Sun, Michael Godfrey, and Ladan Tahvildari—thank you for taking the time to evaluate my work and provide constructive feedback.

I will always cherish the wonderful memories and moments that I shared with my friends and lab mates over the years. You made this experience richer and more enjoyable. I wish you all the success through your journey.

To my beloved parents, Sumanasena and Sakunthala—thank you for your endless love and unwavering support. I am also deeply grateful to my sister (Madhu) and brother (Chathuranga), and to my dear mother-in-law (Harriet), and her family, for the care and encouragement you have given me over the years.

Last but not least, to my husband, Sanka—thank you for being my greatest cheerleader. I am forever grateful.

## **Dedication**

To my beloved husband.

# Table of Contents

Examining Committee	ii
Author's Declaration	iii
Statement of Contributions	iv
Abstract	v
Acknowledgements	vi
Dedication	vii
List of Figures	xiii
List of Tables	xv
List of Abbreviations	xvii
<b>1 Introduction</b>	<b>1</b>
1.1 Problem Statement . . . . .	2
1.2 Thesis Overview . . . . .	3
1.2.1 Tacit Inefficiencies in CI . . . . .	3
1.2.2 Tacit Barriers in CI . . . . .	4

1.3	Thesis Contributions . . . . .	5
1.4	Thesis Organization . . . . .	6
<b>I</b>	<b>Preliminaries</b>	<b>7</b>
<b>2</b>	<b>Background</b>	<b>8</b>
2.1	CI Stakeholders . . . . .	8
2.2	CI Pipeline . . . . .	9
2.3	Chapter Summary . . . . .	10
<b>3</b>	<b>Related Work</b>	<b>11</b>
3.1	Inefficiencies in CI . . . . .	11
3.2	Accelerating CI . . . . .	12
3.3	Barriers in CI . . . . .	14
3.4	Chapter Summary . . . . .	15
<b>II</b>	<b>Tacit Inefficiencies</b>	<b>17</b>
<b>4</b>	<b>CI Timeouts</b>	<b>18</b>
4.1	Introduction . . . . .	18
4.2	Study Design . . . . .	21
4.2.1	(PS) Project Selection . . . . .	21
4.2.2	(DC) Data curation . . . . .	23
4.2.3	(MF) Model Fitting . . . . .	25
4.3	Study Results . . . . .	26
4.3.1	(RQ1) What is the prevalence of CI timeout builds? . . . . .	27
4.3.2	(RQ2) How well can our models explain the incidences of timeout builds? . . . . .	28

4.3.3	(RQ3) What are the most influential features of our models of time-out builds? . . . . .	30
4.3.4	Longitudinal Analysis . . . . .	36
4.3.5	Thematic Analysis . . . . .	39
4.4	Threats to Validity . . . . .	42
4.4.1	Construct Validity . . . . .	42
4.4.2	Internal Validity . . . . .	43
4.4.3	External Validity . . . . .	44
4.5	Practical Implications . . . . .	44
4.6	Chapter Summary . . . . .	46
<b>5</b>	<b>Unused-Dependency Updates</b> . . . . .	<b>47</b>
5.1	Introduction . . . . .	47
5.2	Study Design . . . . .	49
5.2.1	(PS) Project Selection . . . . .	49
5.2.2	(DC) Data Curation . . . . .	51
5.3	Study Results . . . . .	53
5.3.1	(RQ1) What is the prevalence of CI waste due to unused dependencies? . . . . .	53
5.3.2	(RQ2) What are the main sources of CI waste due to unused dependencies? . . . . .	57
5.3.3	Mitigation of CI Waste Due to Updates to Unused Dependencies . . . . .	61
5.4	Threats to Validity . . . . .	66
5.4.1	Construct Validity . . . . .	67
5.4.2	Internal Validity . . . . .	67
5.4.3	External Validity . . . . .	67
5.5	Practical Implications . . . . .	68
5.6	Chapter Summary . . . . .	69

<b>III</b>	<b>Tacit Barriers</b>	<b>71</b>
<b>6</b>	<b>Diversity of DevOps Contributors</b>	<b>72</b>
6.1	Introduction . . . . .	72
6.2	Study Design . . . . .	75
6.2.1	(PS) Project Selection . . . . .	75
6.2.2	(DC) Data Curation . . . . .	78
6.3	Study Results . . . . .	84
6.3.1	(RQ1) Does the perceptible ethnic and gender diversity of DevOps contributors differ from that of non-DevOps contributors? . . . . .	84
6.3.2	(RQ2) How does the distribution of perceptible ethnic and gender diversity change as projects age? . . . . .	91
6.3.3	(RQ3) How does the intersection of perceptible gender and ethnic diversity differ between DevOps and non-DevOps contributors? . . . . .	95
6.4	Threats to Validity . . . . .	100
6.4.1	Construct Validity . . . . .	100
6.4.2	Internal Validity . . . . .	103
6.4.3	External Validity . . . . .	103
6.5	Practical Implications . . . . .	104
6.6	Chapter Summary . . . . .	106
<b>IV</b>	<b>Final Remarks</b>	<b>107</b>
<b>7</b>	<b>Conclusions and Future Work</b>	<b>108</b>
7.1	Contributions and Findings . . . . .	109
7.2	Future Work . . . . .	110
7.2.1	Develop approaches for predicting timeout builds . . . . .	110
7.2.2	Extend the scope of the impact of unused dependencies beyond CI build time . . . . .	110

7.2.3	Mitigate inefficiencies in CI due to smells in CI configurations . . .	110
7.2.4	Explore new business models for CI providers and project maintainers that benefit long-term sustainability . . . . .	111
7.2.5	Explore the challenges of DevOps contributors situated at the intersection of minority groups . . . . .	111
7.2.6	Explore tacit barriers faced by DevOps contributors beyond diversity and inclusion . . . . .	112
<b>References</b>		<b>113</b>

# List of Figures

1.1	An overview of the thesis. . . . .	3
2.1	An overview of a typical scenario where a CI build is triggered and the feedback is sent to the developer. . . . .	9
4.1	An overview of our study design. . . . .	21
4.2	The distributions of the duration of timeout and signal-generating builds. . . . .	28
4.3	Build duration vs. the probability of timing out. . . . .	32
4.4	Timeout ratio vs. the probability of builds timing out. . . . .	34
4.5	An example of builds timeline. . . . .	36
4.6	The number and duration of a timeout cluster. . . . .	37
4.7	The distance and time difference between an isolated timeout and the closest timeout. . . . .	38
5.1	An overview of our study design. . . . .	50
5.2	CI build time consumption of top six projects per month due to unused-dependency commits. The graphs corresponding to other projects are provided in our online appendix. <sup>53</sup> . . . . .	55
5.3	An overview of the DEP-SCIMITAR workflow. . . . .	62
5.4	Distribution of the wasted CI build time of the top six projects before and after applying our tool. . . . .	65
6.1	An overview of our study design showing project selection (PS) and data curation (DC) steps. . . . .	75

6.2	Threshold plot for the number of builds. . . . .	77
6.3	Threshold plot for the number of commits. . . . .	77
6.4	Threshold plot for the number of contributors. . . . .	78
6.5	Bean plots showing the distribution of percentages of DevOps and non-DevOps contributors from four perceptible ethnicities (i.e., White, API, Hispanic, and Black). The solid lines represent the median percentages, and the dotted lines represent the first and third quantiles. . . . .	87
6.6	Bean plots showing the distribution of percentages of contributors perceived as men and women among DevOps contributors as well as among non-DevOps contributors. The solid lines present the median, and the dotted lines present the first and third quantiles. . . . .	89
6.7	Change in the perceptible ethnic diversity of DevOps contributors over time.	92
6.8	Change in the perceptible ethnic diversity of non-DevOps contributors over time. . . . .	93
6.9	Change in the perceptible gender diversity of DevOps contributors over time.	94
6.10	Change in the perceptible gender diversity of non-DevOps contributors over time. . . . .	95
6.11	Change in the perceptible ethnic diversity of the DevOps contributors who are perceptibly women. . . . .	98
6.12	Change in the perceptible ethnic diversity of the non-DevOps contributors who are perceptibly women. . . . .	99

# List of Tables

4.1	Description and rationale of the selected features. . . . .	24
4.2	Model fitness. . . . .	30
4.3	Importance of families. . . . .	32
4.4	Importance of features. . . . .	33
4.5	The extracted themes for CI timeout builds. . . . .	41
5.1	The prevalence of CI waste from unused dependencies. The table presents the total and wasted number of commits and builds. The table further presents figures for both CI providers and consumers. . . . .	54
5.2	Distribution of unused-dependency commits and corresponding build hours over bots and developers. . . . .	59
5.3	Build hours attributed to unused-dependency commits authored by bots. . . . .	59
5.4	Comparison of the total number of commits and build hours stemming from unused-dependency commits between development and runtime dependencies across all the projects in our dataset. . . . .	60
5.5	Build hours resulting from unused-dependency commits in development dependencies. . . . .	61
6.1	Descriptive statistics of our curated dataset of 450 projects. . . . .	83
6.2	A sample of our curated dataset. . . . .	83
6.3	Results of statistical analysis of ethnic diversity metrics for DevOps and Non-DevOps contributors. . . . .	88
6.4	Results of statistical analysis of gender diversity metrics for DevOps and Non-DevOps contributors. . . . .	90

6.5	Results of the statistical analysis of the percentages of the perceptible ethnicities of perceptibly women DevOps and non-DevOps contributors. . . .	97
-----	--	----

# List of Abbreviations

**2PRACE** Mixed Race [82](#)

**AIAN** American Indian/Alaskan Native [82](#), [83](#), [87](#), [90](#), [96](#)

**API** Asian/Pacific Islander [82](#), [83](#), [86](#), [90](#), [91](#), [96](#), [97](#), [99](#), [105](#), [111](#)

**AUPRC** Area Under Precision-Recall Curve [19](#), [29](#), [30](#)

**AUROC** Area Under Receiver Operating Characteristic Curve [19](#), [29](#), [30](#), [35](#), [45](#), [82](#)

**CI** Continuous Integration [1](#), [8](#), [11](#), [108](#)

**CPU** Central Processing Unit [41](#)

**DoF** Degrees of Freedom [26](#), [29](#), [32](#), [33](#)

**DORA** DevOps Research and Assessment [15](#), [73](#)

**GHA** GitHub Actions [8](#), [9](#), [48](#), [50](#), [53](#), [56](#), [63–65](#), [67](#)

**JSON** JavaScript Object Notation [35](#), [51](#)

**MENA** Middle East and North Africa [100](#), [101](#)

**PR** Pull Request [1](#), [8](#), [9](#), [39](#), [41](#), [44](#), [68](#), [108](#)

**RAM** Random Access Memory [41](#)

**SMOTE** Synthetic Minority Over-sampling TEchnique [23](#)

**STEM** Science, Technology, Engineering, and Mathematics [74](#), [96](#)

**US** United States [82](#), [83](#), [93](#), [100](#), [101](#)

**VCS** Version Control System [9](#)

# Chapter 1

## Introduction

Modern software is developed at a swift pace. At the core of this rapid development is [Continuous Integration \(CI\)](#)—the heartbeat of a software project [41]. Development events of importance, such as commits being submitted to a shared repository and creation of [Pull Requests \(PRs\)](#), could be configured to automatically trigger *CI builds*. A CI build orchestrates a series of tasks, including dependency retrieval, compilation, and testing to validate any changes and maintain code integrity [170, 81, 13, 178]. For instance, a CI build triggered by a commit allows the team members to verify whether the corresponding change set integrates seamlessly with the shared repository [56]. Adoption of CI in software projects is known to improve team productivity [155, 80] and software quality [170, 81, 156]. By continuously integrating change sets and running automated tests, CI provides prompt feedback, allowing project teams to address faulty change sets while preventing them from escalating into more costly problems later in the development cycle.

Despite its benefits, CI presents challenges to project teams, which are twofold: barriers and inefficiencies. Barriers include obstacles that may hinder the successful adoption of CI practices. For example, transitioning to CI practices may require a cultural shift within development teams, and prior studies have identified resistance to change as a key barrier to the adoption and success of CI practices [102, 33]. Furthermore, configuring CI pipelines often requires a specialized skill set that falls under the broader domain of DevOps practices [179], which requires expertise that not all team members may have.

In parallel, CI pipelines become susceptible to inefficiencies as they grow in complexity and scale, which can negatively impact their overall effectiveness. For example, long-running builds are an inefficiency in CI [59]. This is because the longer a build executes, the more CI resources it consumes. While failing CI builds are known to take longer

to complete than those that are passing [59], project teams often restart CI builds after a failure [40, 109]. Such restarts are often driven by flakiness in CI builds, where the same change set intermittently passes or fails, consuming CI resources without producing a consistent outcome [68, 147]. However, restarting CI builds is known to change the outcome of only 42% of failing builds, further exacerbating the waste of CI resources [109].

## 1.1 Problem Statement

Inefficiencies in CI that are discussed in prior work are explicit, where team members are cognizant of them (e.g., the overuse of CI resources due to frequent build restarts [109]); however, not all inefficiencies in CI are immediately apparent. These *tacit inefficiencies* can accrue over time without the awareness of the project team. These include, for instance, unexpected CI build terminations, CI builds triggered due to change sets that do not affect the project’s source code [3], redundant tasks in CI pipelines, and outdated configurations [186], increasing the consumption of CI resources.

Besides inefficiencies in CI, team members who create and maintain CI pipelines [131], i.e., DevOps contributors, encounter a set of barriers. Many of the barriers faced by these contributors, such as the lack of specialized skills [179], are explicitly recognized and tied to the technologies used in CI. Barriers may not be explicitly discussed as specific to CI. We refer to such barriers as *tacit barriers*—those that may not stem explicitly from the technology stack, but rather from social and organizational concerns. Such barriers include a lack of diversity among DevOps contributors and a fear of breaking CI pipelines, which can discourage collaboration on CI artifacts.<sup>1</sup>

In this thesis, we set out to understand the prevalence and characteristics of tacit inefficiencies and barriers in CI and provide strategies to mitigate them. To this end, we leverage historical data from CI pipelines and evaluate the following research hypothesis:

**Hypothesis.** Neglecting tacitly accrued inefficiencies and barriers in CI can have a substantial impact on both CI resources (i.e., build time) and the team members in a project who contribute to CI artifacts. Systematically identifying and characterizing these issues can inform the development of strategies that enhance the overall efficiency of CI pipelines.

<sup>1</sup><https://www.harrisonclarke.com/devops-sre-recruiting-blog/diversity-and-inclusion-in-devops>

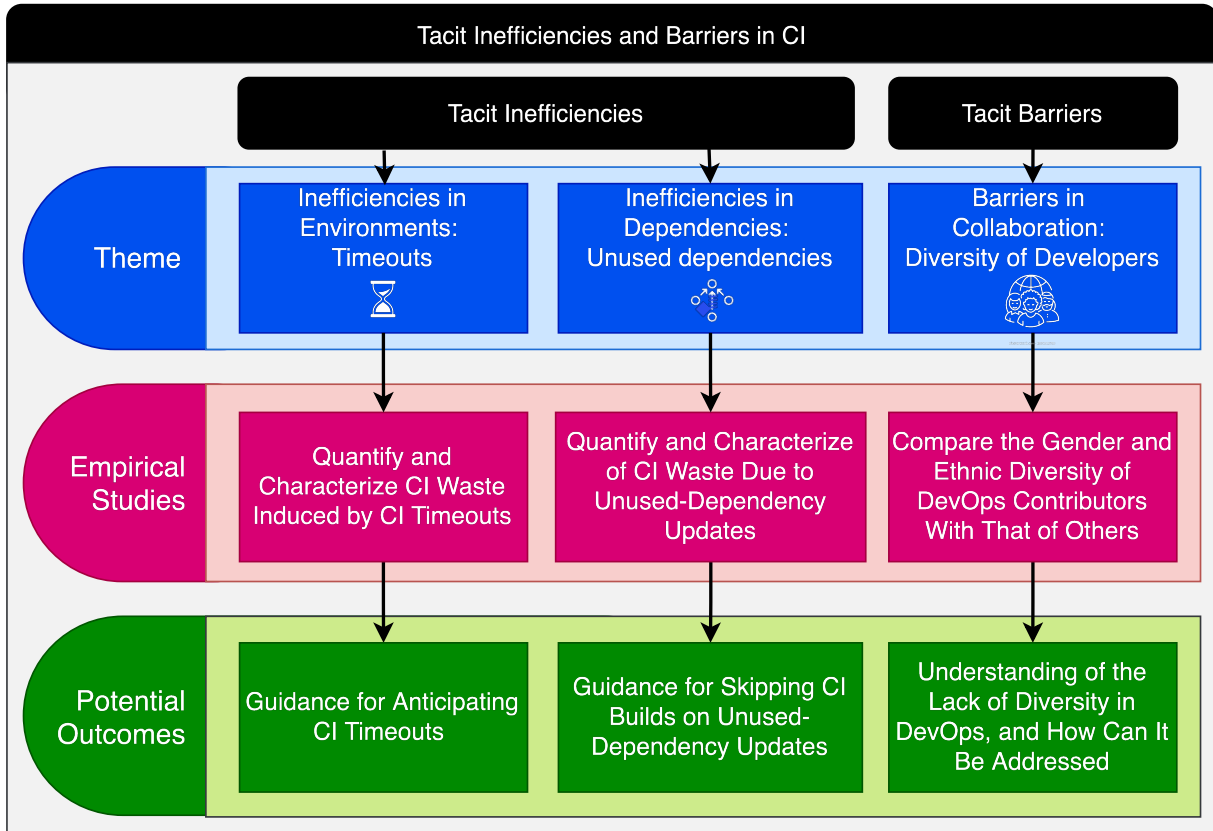


Figure 1.1: An overview of the thesis.

## 1.2 Thesis Overview

In this section, we present an overview of the thesis, with the structure illustrated in Fig.1.1. To evaluate our research hypothesis, we examine tacitly accrued inefficiencies in CI (Section 1.2.1) and tacit barriers in CI (Section 1.2.2).

### 1.2.1 Tacit Inefficiencies in CI

We perform two empirical studies that focus on tacitly accrued inefficiencies in CI: inefficiencies in the underlying CI environment on which CI builds are executed, i.e., CI timeouts, and inefficiencies in dependencies of a project, i.e., unused dependencies.

**Chapter 4 *CI Timeouts*.** CI builds may encounter unforeseen issues in the CI environment, such as network problems<sup>2</sup> and prolonged waiting on external services. To mitigate this, CI providers often impose time limits on builds to prevent erroneous builds from consuming excessive resources in the CI environment. However, when a CI build exceeds this time limit, it is automatically terminated (i.e., timed out),<sup>3</sup> often after consuming a substantial amount of CI resources. While these limits are meant to protect the CI environment from abuse, they can result in the maximum allowable resources being consumed without providing team members with any feedback on whether the CI build would pass or fail [56, 60]. In this chapter, we quantify the prevalence of CI timeouts in open-source projects, and investigate the characteristics that explain the likelihood of CI timeouts.

**Chapter 5 *Unused-Dependency Updates*.** Software projects often rely on external dependencies listed in their configuration files, many of which may not be actively used in the project’s source code [98, 152]. Although these unused dependencies are not essential for building and executing the software, updating their versions in dependency configuration files can still trigger CI builds. Such CI builds are wasteful because they do not impact the actual functionality of the software. Therefore, in this chapter, we quantify and characterize CI waste in open-source projects caused by updates to unused dependencies in order to gain insights into strategies for mitigating this waste.

## 1.2.2 Tacit Barriers in CI

Among the tacit barriers influencing CI, team diversity stands out as particularly important because it has a direct impact on software development [168]. In fact, a report, “Why diversity matters in DevOps,” discusses the potential of diversity in DevOps teams to lead to more streamlined product development.<sup>1</sup> Motivated by this, we explore the role of diversity among team members who contribute to CI artifacts (i.e., DevOps contributors).

**Chapter 6 *Diversity of DevOps Contributors*.** While a lack of diversity in the broader software engineering community has been observed [120, 168], it remains unclear whether this issue persists, worsens, or improves among DevOps contributors. In this chapter, we examine the perceptible ethnic and gender diversity of DevOps contributors compared to other contributors in open-source projects, aiming to better

---

<sup>2</sup><https://github.com/oblador/loki/issues/94>

<sup>3</sup><https://support.circleci.com/hc/en-us/articles/360007188574>

understand the current state and identify opportunities for improving the diversity not only among teams overall but also among DevOps contributors.

## 1.3 Thesis Contributions

The main contributions of this thesis are the following:

- Projects that encounter CI timeouts, a tacit inefficiency in CI, can cause substantial waste of CI resources, posing a burden on CI providers (e.g., CircleCI) due to the amplified impact on infrastructure and service reliability (Chapter 4).
- Project teams and CI providers can use a project’s build history to anticipate CI timeouts because build history is the strongest indicator of potential CI timeouts, and CI timeouts often occur consecutively in clusters (Chapter 4).
- Change sets that are linked to certain files in a project can be more prone to CI timeouts than others. Thus, instead of restarting a new CI build after a timeout, project teams can begin their investigations on the root causes of CI timeouts from these timeout-prone files and fix them (Chapter 4).
- Besides CI builds that time out, those that are triggered by updates to unused dependencies also waste a substantial amount of CI build time. The majority of these wasteful builds are triggered by bots (e.g., Dependabot<sup>4</sup>) that automatically maintain up-to-date versions of dependencies in projects. Such bots need to be optimized to consider actual dependency usage before triggering updates (Chapter 5).
- Project teams may adopt DEP-SCIMITAR<sup>5</sup>, a novel approach that we developed to safely skip CI builds triggered by unused-dependency updates (Chapter 5).
- Tacit barriers to entry, particularly those related to ethnic and gender diversity, are more pronounced among team members who contribute to DevOps artifacts (e.g., CI configuration files) than among others. This disparity calls for a deeper understanding of why DevOps work is less attractive to perceptible minorities (Chapter 6).
- The perceived diversity of both DevOps and non-DevOps contributors is slowly increasing over time, but there is still room for improvement (Chapter 6).

---

<sup>4</sup><https://github.com/dependabot>

<sup>5</sup><https://github.com/nimmiw/dep-scimitar>

- The lack of perceived diversity is amplified when considering the intersection of minority ethnicities and genders among DevOps and non-DevOps contributors. This shows a compounded underrepresentation, indicating the necessity of targeted inclusion strategies (Chapter 6).

## 1.4 Thesis Organization

The remainder of this thesis is structured into four parts: Part I provides the preliminaries of this thesis, i.e., an overview of CI stakeholders and CI pipelines (Chapter 2), followed by a review of related work on inefficiencies and barriers in CI (Chapter 3). Part II explores tacit inefficiencies in CI: CI build timeouts (Chapter 4) and CI builds triggered due to unused dependencies (Chapter 5). Part III analyzes the diversity of members in a project team who contribute to CI-related artifacts, i.e., a tacit barrier in CI (Chapter 6). Part IV presents the final remarks, summarizing the key findings of this thesis and outlining future research directions to reduce inefficiencies and barriers in CI (Chapter 7).

**Part I**

**Preliminaries**

# Chapter 2

## Background

Continuous Integration (CI) is a software development practice where developers frequently merge their change sets into a shared repository, often triggering automated builds to ensure the shared repository remains stable [41, 84]. While CI builds are typically triggered by events such as code commits or PRs, they can also be initiated manually by developers or scheduled to execute at specific times of the day. CI addresses the challenges of late-stage code integration, known as “integration hell,” by enabling continuous feedback and supporting agile and DevOps methodologies [187, 41]. In this chapter, we outline the stakeholders (Section 2.1) and steps in the pipeline of CI (Section 2.2).

### 2.1 CI Stakeholders

There are two primary stakeholder groups in CI: CI consumers and CI providers. CI consumers include project maintainers who oversee the development process, as well as team members who are responsible for designing and maintaining CI configurations. CI providers are companies or services that offer tools and infrastructure for CI, e.g., on-premise CI providers, such as Jenkins,<sup>6</sup> and cloud-based CI providers, such as CircleCI,<sup>7</sup> GitHub Actions (GHA),<sup>8</sup> and Travis CI.<sup>9</sup>

CI consumers may choose a CI provider based on their project-specific requirements. For example, a project maintainer may choose CircleCI if they require a dedicated CI

---

<sup>6</sup><https://www.jenkins.io>

<sup>7</sup><https://circleci.com>

<sup>8</sup><https://github.com/features/actions>

<sup>9</sup><https://www.travis-ci.com/>

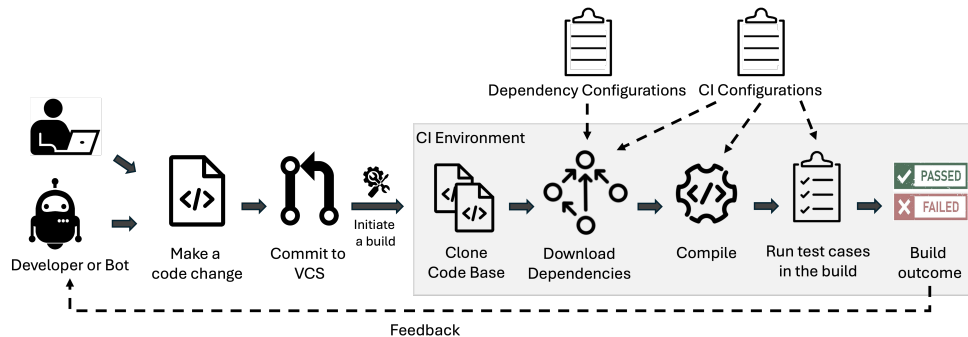


Figure 2.1: An overview of a typical scenario where a CI build is triggered and the feedback is sent to the developer.

provider to build, manage, and run its CI pipelines.<sup>10</sup> On the other hand, a project maintainer may opt for [GHA](#) if they are looking to automate tasks beyond CI, such as automatically generating changelogs when deploying a build in the production environment<sup>11</sup> or automatically assigning change requests submitted by users to team members working on the project.<sup>12</sup>

## 2.2 CI Pipeline

A CI build can be triggered by various events of importance, such as the submission of a change set to a [Version Control System \(VCS\)](#), the initiation of a [PR](#), a scheduled nightly build, or even a manual trigger during a code review. A typical CI build consists of key phases aimed at smoothly integrating change sets and detecting errors early. Fig. 2.1 illustrates an example of a CI build triggered by a commit that submits a change set to the [VCS](#). Note that, in addition to team members, automated bots can also trigger CI builds. The environment in which a CI build is executed may be on-premise or cloud-based infrastructure, including hardware, software, and network resources. For instance, hardware resources may comprise servers for executing tasks and storage systems for archiving build artifacts (e.g., binaries and log files).

<sup>10</sup><https://circleci.com/docs/local-cli>

<sup>11</sup><https://docs.github.com/en/repositories/releasing-projects-on-github/automatically-generated-release-notes>

<sup>12</sup><https://github.com/marketplace/actions/auto-assign-issue>

While the majority of the team members in a project focus on source code files, a subset (i.e., DevOps contributors) takes the responsibility for configuring all the settings and instructions in CI configuration files for executing a CI build. For CircleCI, these are in the `configuration.yml` file within the `.circleci/` directory. GitHub Actions workflows are defined in YML files in the `.github/workflows/` directory, with each file containing multiple jobs to automate various tasks. These configuration files specify which branches or events trigger builds, environment variables, and secure management of secrets. CI configuration files also include the steps to execute a CI build, such as cloning the repository, downloading dependencies from dependency configuration files, compiling the code, and running automated tests. Additionally, they may include quality assurance steps, such as linting to enforce coding standards.

Depending on the outcome of the steps in the CI configuration, a CI build is marked as either **Passed** or **Failed**, indicating a *signal-generating* build that informs the user whether their code changes are successfully integrated [56]. In contrast, *non-signal-generating* builds terminate early due to user aborts, configuration errors, or infrastructure provisioning issues within the CI environment, offering no meaningful feedback to project teams [56].

## 2.3 Chapter Summary

This chapter defines the main concepts of CI. Specifically, we provide an overview of the stakeholders involved in CI, as well as a typical CI pipeline. We also introduce key terms used throughout this thesis, such as CI configurations and dependency configurations. In the next chapter, we survey prior research on CI to situate our empirical studies of tacitly accrued inefficiencies and barriers within the broader context of the existing literature.

# Chapter 3

## Related Work

CI offers several benefits, making it an essential practice in modern software development [14, 170, 81, 157, 46, 54, 187]. The main advantage of CI is the timely feedback it provides on change sets, allowing developers to address defects promptly [54, 46]. This reduces the risk of costly bugs propagating to later stages of the development cycle [81]. CI also enhances team collaboration by minimizing integration conflicts [81, 157], and it increases development velocity through automation, enabling faster delivery of features and updates [81].

While CI offers benefits, it is not without limitations. In this section, we review prior studies most relevant to this thesis. We discuss inefficiencies in CI (Section 3.1), existing solutions to address them (Section 3.2), and barriers in CI (Section 3.3), while positioning our empirical work within the context of existing research.

### 3.1 Inefficiencies in CI

Several studies have investigated inefficiencies in CI, including restarted builds, long-running builds, and unused dependencies, which we detail below.

**Restarted CI Builds.** Previous studies have investigated restarted CI builds, highlighting them as a substantial drain on CI resources [122, 109, 40]. For example, Olewicki et al. [122] analyzed 11,400 commits across six industrial projects, and found that 31% of those commits had at least one restarted build. Maipradit et al. [109] analyzed 66,932 code reviews on OpenStack, and found that 55% of failing CI builds were restarted during code reviews, and the build outcomes changed in only 42% of those builds. Such restarts

are often invoked due to flakiness in builds, and several studies propose tools to mitigate flakiness [68, 4, 147]. For example, Shi et al. [147] proposed a tool, IFIXFLAKIES, to mitigate the flakiness in CI builds due to order-dependent test cases.

**Long-Duration CI Builds.** CI build durations may increase as projects mature, leading to inefficiencies [59, 55, 56]. Ghaleb et al. [59] analyzed 104,442 CI builds from 67 open-source projects, and found that builds that are restarted multiple times are prone to long durations. They further identified CI timeouts as a common consequence of long-duration builds, which is supported by other studies as well [56, 60]. Gallaba et al. [56] examined 23.2 million CircleCI builds across 7,795 projects, identifying 17,917 builds that timed out. These timeouts waste resources and fail to provide useful feedback. Understanding the characteristics of these CI timeouts that distinguish them from other builds would shed light on targeted optimization strategies, which we explore in Chapter 4.

**Inefficiencies in Project Dependencies on CI.** Besides the inefficiencies in CI pipelines, such as restarting and long-duration builds, the inefficiencies in the dependencies used in projects can waste CI resources. For example, having unused dependencies [98, 152, 83] listed in a project’s dependency configuration file is an inefficiency in dependencies that impact CI pipelines. Even though caching dependencies can reduce the CI build time spent on installing unused dependencies [186], maintaining up-to-date versions of those dependencies may still incur a waste of CI build time. For example, Dependabot<sup>4</sup> updates versions of dependencies in projects without considering whether those dependencies are used, often triggering CI builds. Such CI builds waste CI build time because the outcomes of those builds do not impact the project deliverables, as those dependencies are never used. In Chapter 5, we perform a detailed analysis of how unused dependencies impact CI pipelines.

## 3.2 Accelerating CI

Prior studies had proposed approaches to mitigate inefficiencies in CI by accelerating CI builds. Below, we detail these approaches and describe how this thesis builds on them.

**Incremental builds.** By incrementally building only what is affected by change sets, the artifact-building phase of CI can be accelerated. Google’s Bazel,<sup>13</sup> Facebook’s Buck,<sup>14</sup> and Microsoft’s CloudBuild [47] are examples of incremental build systems. However, these systems rely on manually specified build dependency graphs, which can drift out of sync or

---

<sup>13</sup><https://bazel.build>

<sup>14</sup><https://buck.build>

be inaccurately specified [55, 107, 111, 112]. Adopting incremental builds may also require replacing existing build systems, creating a barrier to entry [55] (which we discuss in detail in the next section). Nonetheless, incrementally building is currently limited to only the artifact-building phase in the CI pipeline.

**Caching CI builds.** To address the limitations of incremental builds, several approaches are available to cache CI builds. Gallaba et al. [55] proposed KOTINOS, a build acceleration approach that is agnostic to CI providers. KOTINOS infers dependencies by tracing system calls and testing operations during an initial build. This inferred dependency graph is then used to accelerate future builds by caching the environment in a Docker container and skipping build steps or tests that are unaffected by recent changes. One limitation of this method is that the entire cache would invalidate if the CI environment changes in the later builds (e.g., due to updates in dependency versions), and thus, the cold build procedure will be re-executed (i.e., a fresh image will be created and stored). CI providers also offer in-built support for partial caching of CI builds, such as caching dependencies after retrieval.<sup>15</sup>

**Skipping CI builds.** To save CI build time, prior work also investigated approaches to skip CI builds [55, 3, 2, 86, 87]. For example, Abdalkareem et al. [3] examined 1,813 commits where developers purposely skipped CI builds. This analysis revealed reasons for skipping CI builds (e.g., when change sets only modify non-compileable files, such as documentation). For these reasons, the study proposed a rule-based method (CI-SKIPPER) that automatically identifies commits that could be skipped by CI pipelines.

Other studies proposed methods to predict CI build outcomes not only to enable early feedback on potential build failures, but also to provide the option of skipping builds that are unlikely to fail [180, 76, 25, 138, 85, 159]. For example, Chen et al. [25] proposed BUILDFAST—an approach to predict CI build outcomes by using the change set and history-aware features. Their predictor outperformed the state-of-the-art approaches, achieving a 47.5% improvement in the F1-score for failed builds.

**Skipping test cases.** Other studies proposed approaches to cut down on CI build time by skipping test cases [62, 79, 106]. For instance, Gligoric et al. [62] proposed a method to skip test cases that cannot reach the changed files by tracking their dynamic dependencies on source files. In a new CI build, if none of the source files a particular test case depends on have changed, that test case can be safely skipped.

The above build acceleration methods focus on speeding up builds rather than reducing structural inefficiencies in CI. Incremental builds and caching methods primarily focus on

---

<sup>15</sup><https://docs.github.com/en/actions/writing-workflows/choosing-what-your-workflow-does/caching-dependencies-to-speed-up-workflows>

performance optimization by reducing redundant computations or re-installations, rather than addressing the underlying inefficiencies that silently accrue in CI pipelines. Thus, they do not necessarily prevent waste caused by unnecessary builds. Moreover, although skipping CI builds or test cases can reduce CI build time, such methods do not account for inefficiencies caused by timeouts or updates to unused dependencies. In this thesis, we aim to scrutinize the insights gained from analyzing tacitly accrued inefficiencies in CI, focusing specifically on CI timeouts (Chapter 4) and unused-dependency updates (Chapter 5).

### 3.3 Barriers in CI

In this section, we discuss the barriers faced by contributors who contribute to DevOps artifacts (e.g., CI configuration files) and organizations that adopt DevOps practices.

**Complicated Infrastructure.** The complexity of infrastructure is a frequently discussed barrier to adopting DevOps practices [26, 102, 123, 96]. Indeed, the infrastructure to execute CI builds may be composed of cloud technologies, container orchestrating systems, databases, and logging systems. Claps et al. [26] interviewed ten employees at Atlassian and observed that even minor changes in one component of the CI pipeline often required updates across multiple interconnected systems. Thus, they emphasized the importance of rigorous testing, especially for tightly coupled components like databases, to prevent unintended consequences. The study also highlighted the need for a substantial hardware capacity and special software resources to support robust CI. This complexity is further exacerbated when projects rely on legacy infrastructure that lacks compatibility with modern CI tools and practices, creating additional barriers to automation and scalability [92].

**Lack of Expertise.** An understanding of the infrastructure and its configuration is essential for the successful adoption of CI and related DevOps technologies; however, prior studies have identified the lack of expertise in automation as a major challenge for implementing DevOps practices [183, 105, 93, 146, 26]. For example, Yiran et al. [183] surveyed DevOps practitioners, and found that organizations often lack team members with the necessary expertise to implement CI pipelines effectively. Senapathi et al. [146], as a result of interviewing six experienced developers, reported that a lack of expertise was evident both when upskilling existing team members and during recruitment.

**Tacit Barriers.** In addition to the technical barriers above, DevOps contributors also face several tacit barriers that do not explicitly stem from the technology stack [103, 102, 146, 26, 39, 181]. These barriers arise due to social, cultural, and organizational factors. For example, a lack of motivation is a commonly observed social barrier to adopting DevOps practices [102, 146]. Leppänen et al. [102] conducted interviews with employees from

15 organizations, and found that these organizations are often reluctant to abandon their existing workflows. Others [26, 146] found that the shifting of team roles and responsibilities during the adoption of DevOps practices can lead to ambiguity, causing certain responsibilities to be overlooked.

The aforementioned technical and tacit barriers suggest that DevOps contributors face a unique set of challenges compared to other team members in software projects. Thus, these unique challenges may contribute to a lack of diversity among DevOps developers—a special tacit barrier that is not often discussed in the context of DevOps.

**Lack of Diversity.** While diversity is known to benefit software teams overall, a lack of diversity can itself act as a barrier [18, 23, 162, 168, 124]. In the context of DevOps, annual surveys raise a concern about a lack of diversity among DevOps contributors. For example, *DevOps Research and Assessment (DORA)*'s *Accelerate State of DevOps Report (2021)* [150] highlighted that the percentage of DevOps contributors who found themselves underrepresented has increased from 13.7% in 2019 to 17% in 2021. Furthermore, *DORA*'s *Accelerate State of DevOps Report (2024)* [34] showed that only 12% of DevOps contributors identified themselves as women. Other surveys conducted by Stack Overflow and demographic studies like Zippia's<sup>16</sup> also revealed that men significantly outnumber women in DevOps roles [181]. For instance, the *Stack Overflow Developer Survey (2022)*<sup>17</sup> showed that 94.37% of developers identifying as DevOps contributors were men. Note that recent Stack Overflow surveys from 2023 and 2024 did not contain detailed demographic questions. While these prior analyses and reports provide insights into the overall diversity of the DevOps community, Chapter 6 of this thesis complements them with an empirical analysis of perceptible ethnic and gender diversity at the project level and over time. Moreover, Chapter 6 compares DevOps contributors with non-DevOps contributors within the same set of studied projects.

### 3.4 Chapter Summary

In this chapter, we survey prior research on inefficiencies and barriers in CI. While related work supports our hypothesis that CI is susceptible to such challenges, it remains unclear: (1) what the impact of tacit inefficiencies and tacit barriers in CI is, and (2) where optimization efforts should be focused to streamline the software development process.

---

<sup>16</sup><https://www.zippia.com/devops-engineer-jobs/demographics>

<sup>17</sup><https://survey.stackoverflow.co/2022>

In the remainder of this thesis, we present our empirical studies aimed to address this gap in the literature. In the next chapter, we begin by examining how tacit inefficiencies in the CI environment can affect overall CI performance.

## Part II

# Tacit Inefficiencies

# Chapter 4

## CI Timeouts

**Note.** An extensive version of this chapter appears in the Transactions on Software Engineering journal [174].

### 4.1 Introduction

Inefficient or poorly tuned CI configurations can delay feedback and waste CI resources [171, 58, 184], prompting CI providers such as CircleCI<sup>7</sup> to impose time limits on CI builds.<sup>18</sup> Builds exceeding these limits are terminated to prevent resource abuse, such as from infinite loops or network issues.

A CI build that times out is wasteful for both developers and CI providers. For developers, such builds may produce partial results (e.g., logs or partial test outputs), but they often fail to provide a definitive signal about whether change sets integrate successfully, which can introduce uncertainty and slow down development progress.<sup>19</sup> Addressing CI timeout builds may require adjusting the time limits for timing out. In CircleCI, the default time limit is ten minutes,<sup>20</sup> and project teams can increase this limit in the `.circleci/config.yml` file (see the description about CI configurations in Section 2.2). Properly setting these time constraints is essential to prevent prolonged, resource-intensive build processes. On the other hand, from a CI provider’s standpoint, timeout builds can

---

<sup>18</sup><https://support.circleci.com/hc/en-us/articles/360045268074>

<sup>19</sup><https://github.com/Homebrew/discussions/discussions/4075>

<sup>20</sup><https://support.circleci.com/hc/en-us/articles/16616033407131-Max-Runtime-in-CircleCI-Server>

result in throughput degradation, as long-running or stalled builds can reduce overall system responsiveness. Moreover, persistent inefficiencies can lead to customer dissatisfaction. Therefore, timeout builds can be costly in terms of service quality, infrastructure utilization, and long-term sustainability. Below are our RQs and a preview of the corresponding answers:

**(RQ1) What is the prevalence of CI timeout builds?** By quantifying how widespread timeout builds are across projects, we can better understand the scale and severity of the problem. In this RQ, we aim to estimate the prevalence of CI timeouts by calculating their frequency and comparing the durations of CI timeout builds to other signal-generating builds in the dataset curated by Gallaba et al. [56].

**Results.** Out of 7,795 projects, 936 (12%) have at least one timeout build. The median number of timeout builds among these projects is four; however, 10% of the projects have 44 or more timeouts each, while an extreme 4% have 100 or more timeouts each, indicating a highly skewed distribution [129]. We further find that the median timeout duration is 19.7 minutes, nearly five times longer than the median duration of signal-generating builds. This suggests that CI timeout builds are a substantial issue for GitHub projects.

**(RQ2) How well can our models explain the incidences of CI timeout builds?** Identifying the characteristics of CI builds that are most indicative of timeouts is valuable for informing strategies to reduce their occurrence. To this end, we fit a statistical regression model using a set of families of features extracted from CI builds to classify timeout builds from other types of builds. In this RQ, we aim to determine whether such a model can effectively distinguish between builds that time out and those that execute until completion. In particular, we assess our model’s fitness by evaluating its discriminatory power, i.e., [Area Under Receiver Operating Characteristic Curve \(AUROC\)](#), calibration of risk estimates (i.e., [Brier Score](#) [19]), stability (i.e., [bootstrap-calculated optimism](#) [43]), and ability to balance precision and recall, i.e., [Area Under Precision-Recall Curve \(AUPRC\)](#).

**Results.** Our statistical model achieves an [AUROC](#) of 0.939—a discriminatory power that vastly surpasses that of naïve baselines, such as random guessing ([AUROC](#) of 0.5). Moreover, the model achieves a Brier score of 0.008—a calibration score that suggests the risk estimates of the model are highly reliable. In terms of stability, our model has only a small optimism penalty of less than one percentage point in terms of both [AUROC](#) and Brier score, suggesting that the model is unlikely to be overfitted to the data on which it is trained. Lastly, our model achieves an [AUPRC](#) of 0.319, outperforming the baseline [AUPRC](#) of 0.012 [139].

**(RQ3) What are the most influential features of our models of CI timeout builds?** This RQ seeks to use our statistical model to identify the characteristics that best explain CI timeouts. To assess the explanatory power of each family of features, we perform Wald  $\chi^2$  (a.k.a., “chunk”) tests [136].

**Results.** We find that CI timeout builds are best characterized by the family of build history features, which contains the status of recent builds, their durations, and the proportion of the builds that have previously timed out. Indeed, build history features contribute to more than half (70%) of the explanatory power of the model. Furthermore, our model reveals that timeout tendency features (e.g., file tendency) are among the most powerful features to influence timeout builds, suggesting that change sets that include specific files or components are more often implicated in CI builds that time out than in others. We further find that a non-negligible proportion (15%) of files more often appears in the change sets of timeout builds than signal-generating builds.

Overall, our results highlight the importance of considering the historical context of builds for CI service providers and consumers to anticipate timeout builds. For example, CI providers could target efforts to enhance their infrastructure, optimize resource allocation, and fine-tune their systems to cater to specific demands of individual software projects, reducing the occurrence of timeouts. CI consumers can also leverage this knowledge to prioritize their attention to recent builds and their associated characteristics. Moreover, team members may identify timeout-prone files and focus their efforts on those files when troubleshooting CI timeouts.

Inspired by the above results, we perform a longitudinal analysis of the occurrences of timeout builds. We find that the majority (64.03%) of the timeout builds occur in clusters (i.e., timeout builds tend to occur consecutively). A considerable proportion (20%) of these clusters comprise at least six builds. We also find that once a cluster of timeout builds is observed, it takes a median of 24 hours before a signal-generating build occurs. This suggests the existence of underlying systemic issues that contribute to prolonged periods of consecutive builds that time out, and to further understand the root causes of build timeouts, we conduct a thematic analysis of 79 GitHub discussions, encompassing 406 comments. This analysis uncovers six primary causes for CI timeouts. Among these, the most frequent root cause is inefficiencies in testing processes.

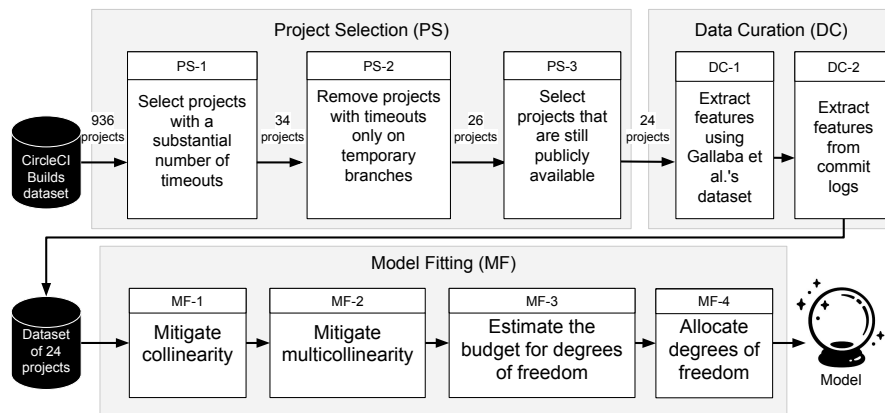


Figure 4.1: An overview of our study design.

## 4.2 Study Design

In this section, we describe our procedures for selecting projects (PS), curating our dataset (DC), and fitting our models (MF) to characterize timeout builds. Fig. 4.1 shows an overview of our study design.

### 4.2.1 (PS) Project Selection

For our analysis, we require a large and rich collection of build records from a realistic CI service. Since there has been an increase in the adoption of CI practices [63], a proliferation and diversity of CI services exist in the market to cater to increasing demands of collaborative software development and DevOps practices. As indicated in prior work [56], CircleCI is one of the most popular CI service(s). In fact, CircleCI has 748k installations over an eight-year period [56]. Given its widespread usage, we focus our analysis on CircleCI. Thus, we begin with the Gallaba et al. [56] dataset of CircleCI builds which explicitly labels passing builds, failing builds, and timeout builds that we need for our study. It is important to highlight that a substantial portion of the timeout builds in the dataset are contributed by specific projects. Additionally, GitHub accommodates projects that are still not yet mature. Conducting a closer examination of mature and disproportionately affected projects would yield valuable insights regarding the workload for the CI provider and raise awareness among project maintainers. To ensure a focused analysis isolating the key factors contributing to these issues, we apply the three-step filtering criteria in Fig. 4.1 (PS) to the projects from the original dataset. Below, we describe each step.

**(PS-1) Select projects with a substantial number of timeout builds.** Our initial dataset contains 936 projects that suffer from at least one timeout. To conduct our study on a set of projects that suffer from a meaningful number of timeouts, we sort the projects in descending order of the timeout frequency. Note that our study requires in-depth data extraction (e.g., collecting the commit and file records for each project). Therefore, to make the study feasible, we consider a subset of projects that account for more than half of all timeout builds collected from the entire dataset. This step yields 34 projects covering 54% of all timeout builds collected from the entire dataset. The selected projects have a rich development history with a median of 3,966 commits made by a median of 82 contributors. Moreover, this dataset includes popular projects, such as the palantir/atlasdb<sup>21</sup> project.

**(PS-2) Remove projects having timeout builds only on temporary branches.** For each surviving project, we select the timeout builds that appear on its main/master branch. Doing so ensures that all selected commits corresponding to the builds are still accessible, mitigating inconsistencies caused by the removal of temporary branches. We find eight of the selected projects exclusively contain timeout builds on temporary branches, and we filter these projects out of further analysis. We find 26 projects that survive this step.

**(PS-3) Select projects that are publicly available.** We need to extract project-related data, such as commits and lines of code. We select projects that are publicly available when conducting this work (22<sup>nd</sup> November 2022). We use the GitHub API to check the accessibility of projects. Doing so excludes two projects, leaving us with 24 projects eligible for our analysis.

Our final dataset consists of 105,663 CI builds spanning 24 projects. Of these builds, 1,301 are timeouts, while the remaining builds generate a pass/fail signal. Note that our dataset is imbalanced in terms of the two classes—a common phenomenon in software engineering research [9]. Class balancing is particularly important in scenarios where the minority class is of greater interest than the majority, such as in fraud detection or rare disease diagnosis [77]. Class balancing tends to improve the recall (by fitting the model in an environment where the minority class is more prevalent than it actually is, a model with well balanced classes is likely to raise timeout alerts and catch timeout examples), but at the cost of precision (since the model is prone to raising plenty of timeout warnings, in real-world scenarios, where timeouts are rare, the model is likely to raise plenty of false alarms). In our case, although the minority class (timeouts) is of great importance, we refrain from applying class balancing to avoid inflating the false positive rate. Indeed, we are interested in modelling the characteristics of builds that are most likely timeouts

---

<sup>21</sup><https://github.com/palantir/atlasdb>

and not the ones that have a little chance of being a timeout; thus, leaving the classes imbalanced in our dataset creates a model that is reflective of real-world scenarios. That said, to see if there is a substantial change in the results after balancing the classes, we rerun our experiments separately in the following class balancing scenarios: (1) Oversampling with the [Synthetic Minority Over-sampling TEchnique \(SMOTE\)](#) [24], (2) Undersampling with the Random method [75, 134], and (3) Combining both [SMOTE](#) oversampling and random undersampling. We do not observe substantial changes in our results. Thus, we use the dataset without any class balancing in the rest of this study. A detailed description of this analysis is available in our Online Appendix B.<sup>22</sup>

Furthermore, we find that the percentage of timeouts varies from one project to another. For example, the `docker-atlassian-confluence/cptactionhank` project accounts for 20.8% of the timeouts in our sample, whereas the median proportion across the 24 projects is 2.3%. To account for the impact of any bias introduced by this project, we rebuild our statistical model, which we discuss in Section 4.2.3 (MF); we do not observe any substantial differences in the model fits. Interested readers can refer to our Online Appendices C and D<sup>22</sup> for details.

## 4.2.2 (DC) Data curation

To determine a set of features that characterize timeout builds, we consult the related literature in the areas of build outcome prediction [138, 25, 125] and defect prediction [161]. Table 4.1 shows the initial list of 19 features that span five properties of a build, along with the rationale for each feature’s impact in the context of build outcomes. We use Gallaba et al.’s dataset [56] and Git commit logs to extract these features. Fig. 4.1 (DC) shows an overview of this step, which we describe in detail below.

**(DC-1) Extract features using Gallaba et al.’s dataset [56].** For each build in our dataset, we extract build-history features, i.e., the most recent build outcome, the most recent build duration, and the timeout ratio, i.e., the ratio of the total number of timeout builds in a project to the total number of builds of the project that triggered before the build under analysis. We also extract queuing-related features, such as the build queued time (the month, day, hour, and minute). We also extract features concerning the tendency of builds to timeout [161], such as the number of previous timeout builds that are linked to the commits by the same authors (author tendency) and the number of prior timeout builds with changes to the same files (file tendency). Build history and tendency features may not be precise at the beginning but will adapt to more accurately reflect values over time.

---

<sup>22</sup><https://doi.org/10.5281/zenodo.10901318>

Table 4.1: Description and rationale of the selected features.

Family	Description/Rationale
Build history	<b>Recent build status:</b> Whether the previous build timed out or not. <u>Rationale:</u> If a project has a recent history of timeout builds, the build is more likely to time out (inspired by the prior work [138, 25]).
	<b>Recent build duration:</b> The duration of the prior build. <u>Rationale:</u> A project with a recent history of long build durations may be more likely to have the build take longer and be timed out (inspired by GitHub issues <sup>23</sup> ).
	<b>Timeout ratio:</b> The proportion of builds that timed out. <u>Rationale:</u> Inspired by studies that predict build outcomes [25], we expect greater timeout ratios to portend future timeouts.
When	<b>Queued month, day, hour, and minute:</b> The moment when the build was queue for processing. <u>Rationale:</u> Build requests of particular times may be more/less prone to timeouts.
Size	<b>LOC:</b> The number of lines of code within the files changed. <u>Rationale:</u> The size of the files changed may have an impact on the likelihood of the build to time out.
	<b>Insertions, deletions, and files:</b> The sum of inserted/deleted lines of code and the number of unique files touched. <u>Rationale:</u> The size of the change corresponding to the build may have a relation to timeout builds (inspired by similar studies on build failure prediction [138, 125]).
Author experience	<b>Changes to related files/changes to any file:</b> The sum of prior changes by the authors to (a) the files changed; and (b) any file. <b>Lines added to/deleted from related files/any file:</b> The sum of prior lines of code added/deleted by the author to (a) the files changes; and (b) any file. <u>Rationale:</u> The familiarity of developers with the overall codebase and specific areas may have an impact on the timeout builds [125].
Timeout tendency	<b>Author tendency:</b> The number of prior timeout builds that contain commits by the authors. <u>Rationale:</u> If the author tendency of a build is high, it has a high chance of timing out [161].
	<b>File tendency:</b> The number of prior timeout builds that contain changes to the files. <u>Rationale:</u> We hypothesize that the higher the file tendency of a certain build, the higher the chance of that build timing out [161].

**(DC-2) Extract features from commit logs.** To get the change set size-related data [125], we analyze the commit log of each project, and extract the unique number of files added, modified, or deleted, as well as the number of lines inserted or deleted in the commits associated with each build. We also extract the features that estimate the project-specific experience of the authors of the commits [125]. For instance, we derive features like the total number of prior commits made by the commit authors associated with each build.

### 4.2.3 (MF) Model Fitting

Our goal is to study what characterizes timeout builds. In this study, we use statistical regression models, which, unlike other classification techniques, emphasize interpretability. In fact, statistical models like logistic regression provide clear insights into how different factors influence outcomes, making them ideal for nuanced analysis.<sup>24</sup> Thus, we fit logistic regression models using the approach recommended by Harrell Jr. [72]. This approach relaxes linearity assumptions using restricted cubic splines to allow features to share complex relations with the outcome (i.e., the likelihood of inducing a timeout in our case). Fig. 4.1 (MF) provides an overview of the steps we follow.

**(MF-1) Mitigate collinearity** Collinear features distort each others' importance in the model [114, 113]. Thus, we first check for collinearity among our features using Spearman's  $\rho$  rank correlation [154]. We choose a rank correlation instead of other types of correlation measures (e.g., Pearson correlation coefficient) because the rank correlation can detect nonlinear correlations. Similar to prior work [160, 64, 59], we use  $\rho = 0.7$  as our threshold, i.e., any pair of features with  $\rho > 0.7$  should have one of the features removed prior to model interpretation.

The hierarchical overview of the correlations among the features is shown in Fig. E.1 in our Online Appendix E.<sup>22</sup> Selecting one feature from the pairs of features that have  $\rho > 0.7$  eliminates the following six features: loc, files, lines added to the related files, lines added to any file, lines deleted from the related files, and lines deleted from any file. We provide details on reasons for choosing one over the other in our Online Appendix E.<sup>22</sup> This step retained 13 non-correlated features.

**(MF-2) Mitigate multicollinearity.** We perform a redundancy analysis on the surviving 13 non-correlated features to mitigate multicollinearity that can introduce noise in model interpretation. A feature may introduce multicollinearity if it can be modeled using

---

<sup>24</sup><https://www.fharrell.com/post/stat-ml>

the other features. We eliminate such redundant features using the `redun` function in R, which fits a set of 13 models that each explain one feature using the 12 other features. Features having model fits that exceed the threshold ( $R^2 > 0.9$ ) are recommended for exclusion [69]. Applying `redun` to our set of features did not identify any additional features for exclusion. Note that reducing collinearity and multicollinearity helps to identify the independent contributions of each feature to the outcome [99].

**(MF-3) Estimate the budget for Degrees of Freedom (DoF)** [43, 70]. All of the features are allocated at least one DoF in our fit. A feature that is allocated a single DoF can only capture monotonic and linear relationships with the likelihood of a CI build timing out. Allocating additional DoF to features allows our model to capture nonmonotonic and nonlinear relationships with the likelihood of a CI build timing out [44]. On the other hand, spending additional DoF to fit our model increases its risk of overfitting (i.e., being too specifically tuned to the training data to apply to testing examples). This tradeoff between model expressiveness and the risk of overfitting is often balanced by respecting a DoF budget [71]. Following prior work [72, 73], the DoF budget for a logistic regression model can be estimated as  $\frac{n}{15}$ , where  $n$  is the number of records in the minority class. Thus, the DoF budget for our model is  $\frac{1,301}{15} = 86$ .

**(MF-4) Allocate DoF.** To expend our budget prudently, we assign more DoF to features that are most likely to have a nonmonotonic relationship with CI timeouts, as determined by Spearman’s multiple  $\rho^2$ . Fig. F.1 in our Online Appendix F<sup>22</sup> shows the  $\rho^2$  value for each feature. From the figure, we observe that the recent build status, timeout ratio, author tendency, file tendency, and recent build duration have higher  $\rho^2$  values than the other features. Thus, we allocate three DoF for those features except for the status of the most recent build since it is a binary feature.

Finally, we fit our regression model to our data, applying restricted cubic splines [71] to the features with additional DoF. These splines smooth transitions between direction changes using cubic fits, while allowing tail regions to retain more linear (straight) shapes. We make our dataset and the replication package available online.<sup>22</sup>

## 4.3 Study Results

In this section, we describe the approach to answer our research questions and the corresponding results.

### 4.3.1 (RQ1) What is the prevalence of CI timeout builds?

To understand the prevalence of CI timeouts, below, we report on our exploratory analysis of timeouts in CircleCI. In particular, we analyze the frequency at which timeout builds occur and the quantity of waste that timeouts generate.

## Approach

We begin with a set of 936 projects that contain at least one timeout build from the dataset curated by Gallaba et al. [56], which includes CircleCI builds from 7,795 open-source GitHub projects; see Section 4.2 (PS-1). This dataset contains the outcome (e.g., pass, failed, timeout, and canceled) for each of the CI builds that it contains. For our analysis, we select the projects that have at least one CI build, with the outcome status being “timeout.” We use this dataset to analyze the distribution of CI timeouts across these 936 projects and to compare the duration of timeout builds to that of other CI builds.

## Results

Below, we report the results of our analysis.

**Observation 4(1): The distribution of CI timeouts in our dataset is skewed.** Among the projects with at least one timeout, a median of four timeout builds is observed, suggesting that the distribution is skewed [129]. Indeed, we find that 10% of the projects account for 44 or more timeouts each, while an extreme 4% of the projects account for 100 or more timeouts each. Overall, this suggests that timeout builds are a relevant issue in the context of GitHub projects.

**Observation 4(2): The median duration of a CI timeout build is fivefold longer than that of other builds.** Fig. 4.2 shows the durations of signal-generating builds [56] (dark grey) and timeout builds (light grey) in the 936 projects that have at least one timeout. From the figure, we observe that a timeout lasts for a median of 19.7 minutes. This is almost fivefold longer than the median duration of signal-generating builds with fail or pass outcomes (3.4 minutes). In certain situations, the issue of timeouts is exacerbated. For example, in the Homebrew/linuxbrew-core project,<sup>25</sup> the median duration of a timeout build is 125.3 minutes, which is 21 times the median duration of a signal-generating build.

---

<sup>25</sup><https://github.com/Homebrew/linuxbrew-core>

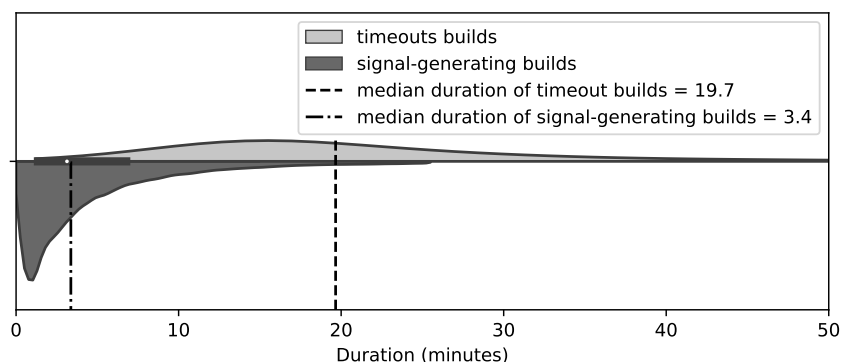


Figure 4.2: The distributions of the duration of timeout and signal-generating builds.

Even the projects that are accountable for small proportions of timeouts in the dataset can be attributable to a large quantity of waste. For example, `coala/coala` project<sup>26</sup> accounts for only 1.7% of the timeouts in the dataset, but it results in a total waste of 111 build hours.

Our results highlight the disproportionate impact of CI timeouts on build time consumption. Even infrequent timeouts can lead to substantial waste of build time, emphasizing the need for mitigation strategies.

### 4.3.2 (RQ2) How well can our models explain the incidences of timeout builds?

Below, we report on the approach used to assess our statistical regression model (trained in Section 4.2.3) for distinguishing CI timeouts from other builds, along with the corresponding results.

#### Approach

We evaluate the fitness of our model according to (a) its discriminatory power, (b) the calibration of its risk estimates, (c) the stability of the fit, and (d) the ability to balance precision and recall, particularly in datasets with imbalanced classes like ours. We estimate

<sup>26</sup><https://github.com/coala/coala>

the discriminatory power of our model using the **AUROC** [69], which plots the true positive rate against the false positive rate; **AUROC** values of 0, 0.5, and 1 represent the worst discrimination, random guessing, and perfect discrimination, respectively. We estimate the calibration of the risk estimates that are produced by our model using the *Brier Score* [19], i.e., the mean squared error of the predicted probabilities. A Brier score of 0 indicates the perfect calibration, whereas a score of 1 indicates the worst. Finally, we estimate the stability of our model fitness using the bootstrap-calculated optimism [43]. We begin by obtaining a sample from our dataset using bootstrap sampling. Then, we refit our logistic regression model to this bootstrap sample with the same allocation of **DoF** used in the original dataset. Next, we calculate the **AUROC** and Brier score of this bootstrap model when (a) reapplied to the bootstrap sample on which it was trained and (b) on the original sample. After that, we estimate the **AUROC** optimism by subtracting the respective fitness measures in the bootstrap sample from that of the original data. We repeat this process for 1,000 bootstrap iterations, and report the mean optimism values. The closer the mean optimism values of these fitness scores are to zero, the greater the stability of the fit. Lastly, we calculate the precision and recall for various threshold values representing the likelihood of a CI build being classified as a “timeout build.” Next, we compute the **AUPRC** to measure our model’s ability to balance precision and recall across different probability thresholds in the context of our imbalanced-class dataset [139]. The **AUPRC** is also a value between 0–1. We then compare our **AUPRC** with a baseline approach. The baseline is determined by the positive class prevalence, i.e.,  $\frac{tp}{tp+tn}$  [139]. The baseline appropriate for a balanced class distribution is 0.5. However, for our dataset, the baseline is  $\frac{1,301}{1,301+104,362} = 0.012$ .

## Results

Table 4.2 shows the results of our model’s fitness, and we make the following observations based on the table.

**Observation 4(3): Our model can discriminate between timeout and non-timeout builds effectively, with well-calibrated risk estimates.** Our model achieves an **AUROC** of 0.939, vastly surpassing the **AUROC** of naïve baselines, such as random guessing (**AUROC** of 0.5). Also, our model achieves a Brier score of 0.008—our model has a near-perfect calibration, and its risk estimates are highly reliable [19].

Table 4.2: Model fitness.

<b>AUROC</b>	<b>Brier Score</b>	<b>AUPRC</b>
0.939	0.008	0.319

**Observation 4(4): Our model is highly stable to bootstrap-simulated variability [72].** The mean optimism value for **AUROC** measure is 0.0001, which is close to perfect optimism [72]. This shows that the **AUROC** value calculated using the bootstrap samples and the **AUROC** value computed using the original dataset are not substantially different. Similarly, the mean optimism value for the Brier score measure is -0.0002. Such small optimism penalties below 1% point suggest that the model is unlikely to be overfitted to the data on which it was trained.

**Observation 4(5): Our model demonstrates a commendable balance between precision and recall.** Our model achieves an **AUPRC** of 0.319, surpassing the corresponding baseline of 0.012. This shows its effectiveness in distinguishing positive instances and minimizing false positives, which is especially crucial in our dataset of CI builds with an imbalanced class distribution of timeout builds and other builds.

Our model demonstrates a strong discriminatory power and calibration (**AUROC** of 0.939 and Brier score of 0.008). Moreover, the fit is highly stable across different bootstrap samples (mean optimism values of 0.0001 and -0.0002 for **AUROC** and Brier score, respectively). Lastly, our model successfully balances precision and recall, achieving an **AUPRC** of 0.319, which surpasses the respective baseline.

### 4.3.3 (RQ3) What are the most influential features of our models of timeout builds?

Below, we report on our approach to determine the most influential features that characterize CI timeout builds and the corresponding results.

#### Approach

First, we estimate the importance of each family of feature(s) using the Wald  $\chi^2$  maximum likelihood (a.k.a., “chunk”) tests [136]. The Wald  $\chi^2$  value indicates whether the model is

statistically different from the same model in the absence of a given independent family of feature(s). The higher the Wald  $\chi^2$  value, the greater the explanatory power of the feature family in identifying timeout builds. While analyzing feature families allows understanding the collective impact of feature families on timeout builds, analyzing individual features allows understanding the specific contribution of each feature to timeout builds. Thus, we analyze the Wald  $\chi^2$  values of individual features as well.

Furthermore, we complement our analysis by plotting response curves<sup>27</sup> for the most important features to analyze the trend of the probability of a CI build timing out as the feature value varies. All other features are held constant at “typical” values, i.e., median for numeric features and mode for categorical features. These plots also show the 95% confidence intervals of the probabilities that are calculated based on 1,000 bootstrap iterations.

## Results

Table 4.3 shows Wald  $\chi^2$  values for each of the families of features of our model. Note that the results are shown in two columns. The *Overall* column shows the explanatory power of all of the degrees of freedom that have been allocated to a family, while the *Nonlinear* column shows the explanatory power that the additional degrees of freedom provide. If no additional degrees of freedom have been allocated to a family, a dash (-) symbol is shown in the nonlinear column. The table shows that the ratio of the Wald  $\chi^2$  of all the nonlinear degrees of freedom to that of the entire model is  $\frac{645.04}{4,350.03} = 0.15$ , indicating that the magnitude of the contribution of additional degrees of freedom to our model is substantial and statistically significant ( $p < 0.001$ ). Moreover, the two families of features (i.e., build history and timeout tendency) that are allocated additional degrees of freedom contribute a significant amount of explanatory power.

**Observation 4(6): The build history family is the most important family for explaining the likelihood of timeout builds.** Table 4.3 shows that the build history family has the highest Wald  $\chi^2$  value. Furthermore, the ratio of the Wald  $\chi^2$  of build history to that of the entire model is  $\frac{3,063.19}{4,350.03} = 0.70$ , i.e., the build history family is the only family that can explain 70% of the variance of our model.

Furthermore, we show the importance of individual features in Table 4.4. The table shows that the recent build status, timeout ratio, and recent build duration are the most important features in the model. This indicates that projects with past timeout builds are

---

<sup>27</sup><https://cran.r-project.org/web/packages/rms/rms.pdf>

Table 4.3: Importance of families.

Family		Overall	Nonlinear
Build history	DoF	5	2
	$\chi^2$	3,063.19 ***	440.14 ***
Timeout tendency	DoF	5	3
	$\chi^2$	20.77 ***	20.27 ***
Queued time	DoF	4	-
	$\chi^2$	6.79 ○	-
Author experience	DoF	2	-
	$\chi^2$	1.95 ○	-
Size	DoF	2	-
	$\chi^2$	0.15 ○	-
<b>Entire model</b> (all families)	DoF	18	5
	$\chi^2$	4,350.03 ***	645.04 ***

○  $p \geq 0.05$ ; \* $p < 0.05$ ; \*\* $p < 0.01$ ; \*\*\* $p < 0.001$

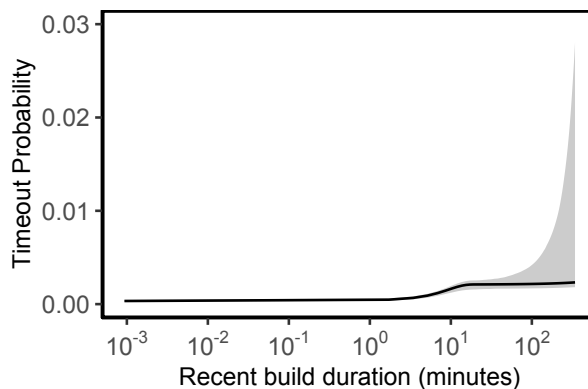


Figure 4.3: Build duration vs. the probability of timing out.

more likely to accrue timeout builds. In fact, CI builds are often restarted in response to timeout builds [40]. Doing so without systematically addressing the cause of such timeouts may result in consecutive timeouts. To better understand the underlying relationship between past and future timeouts, we conduct a longitudinal analysis of the incidences of timeout builds in Section 4.3.4.

Table 4.4: Importance of features.

Family	Feature		Overall	Nonlinear
Build history	recent build	DoF	1	-
	status	$\chi^2$	1,215.49 ***	-
	timeout ratio	DoF	2	1
		$\chi^2$	739.36 ***	287.02 ***
Build history	recent build	DoF	2	1
	duration	$\chi^2$	133.69 ***	128.98 ***
Timeout tendency	author tendency	DoF	3	2
		$\chi^2$	12.16 **	11.74 **
Timeout tendency	file tendency	DoF	2	1
		$\chi^2$	8.89 *	8.22 **
Queued time	queued month	DoF	1	-
		$\chi^2$	2.64 *	-
	queued day	DoF	1	-
		$\chi^2$	2.83 °	-
	queued hour	DoF	1	-
	$\chi^2$	0.84 °	-	
Queued time	queued minute	DoF	1	-
		$\chi^2$	0.55 °	-
	changes to related files	DoF	1	-
Author experience		$\chi^2$	1.67 °	-
	changes to any file	DoF	1	-
Author experience		$\chi^2$	0.42 °	-
	deletions	DoF	1	-
Size		$\chi^2$	0.11 °	-
	insertions	DoF	1	-
		$\chi^2$	0.05 °	-

°  $p \geq 0.05$ ; \* $p < 0.05$ ; \*\* $p < 0.01$ ; \*\*\* $p < 0.001$

The response curves corresponding to build history features are shown in Fig. 4.3 and Fig. 4.4. Fig. 4.3 shows the response curve for the direction of the relationship between the duration of the most recent build and the probability of a CI build timing out. Accordingly, as the duration of the recent build increases, the probability of the current build timing out increases. While this appears to be a weak relationship in the figure (when compared to

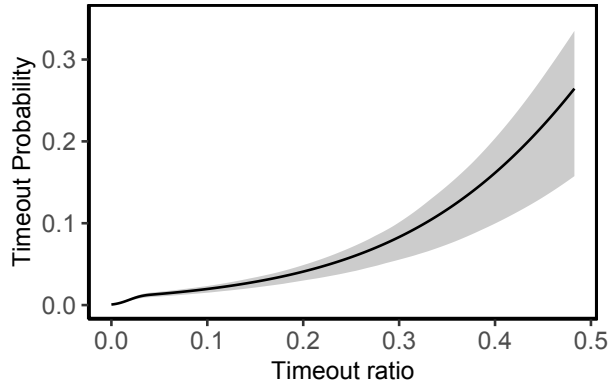


Figure 4.4: Timeout ratio vs. the probability of builds timing out.

the strongest features of our model), the chunk test for this feature, as shown in Table 4.4, yields an explanatory power of 133.69 out of 4,350.03, with a significant p-value ( $< 0.001$ ).

Fig. 4.4 shows that as the timeout ratio increases, the probability of the current build timing out increases exponentially. This suggests that projects that experienced a high proportion of timeouts in the past will likely continue to suffer from a high rate of timeouts in the future. Note that the 95% confidence intervals are narrow for small values of the explanatory features but tend to broaden as the feature values increase. Having broader confidence intervals does not invalidate our model, but is instead a reflection of how the data in our sample supports the trend (i.e., there are fewer samples supporting the broader areas of the curve).

**Observation 4(7): Timeout tendency is the second strongest family in explaining timeout builds.** Table 4.4 shows that both forms of timeout tendency—file tendency and author tendency—have contributed statistically significant amounts of distinct explanatory power. In particular, the importance of file tendency to the model suggests that changes in certain files have a tendency to lead to timeout builds. A similar trend is observed for author tendency as well. To gain a richer perspective on the relationship between timeout builds and file tendency, we inspect the files that are associated with timeout builds. Of these files, we find that 15% appear more in the change sets of timeout builds than signal-generating builds. That is, the number of timeout builds associated with these files (median = 4) is greater than the number of signal-generating builds associated with them (median = 2). For example, `Branch-SDK/src/main/java/io/branch/referral/Defines.java` is one such file in `BranchMetrics/android-branch-deep-linking-attribution`

project.<sup>28</sup> This file had been associated with 136 timeouts, which is three times more than the number of signal-generating builds associated with the file. A close inspection of the content of the file reveals that this file defines [JavaScript Object Notation \(JSON\)](#)<sup>29</sup> keys, request paths, and link parameters. Overall, the file provides an essential way to access and handle various keys used in the system. Furthermore, we find that 24 artifacts in the project rely on those keys. Thus, such a file may have a broad impact on the system when it is changed, which can consequently impact the build outcome.

Lastly, we find that timeout tendency and build history features together account for a significant portion (4,297.41) of the model’s total explanatory power (4,350.03), providing evidence of the gain when fitting the model only by considering strong families of features.

**Observation 4(8): A subset of observed features does not play a significant role in identifying CI builds that are likely to time out.** The families of queued time, author experience, and size do not significantly contribute to the explanatory power of our model compared to the remaining two families. The family of features capturing size has the least importance to the model. Table 4.3 shows Wald  $\chi^2$  values for the aforementioned families; the ratio of the Wald  $\chi^2$  of the family to the entire model is  $\frac{6.79}{4,350.03} = 0.0015$ ,  $\frac{1.95}{4,350.03} = 0.0004$ , and  $\frac{0.15}{4,350.03} = 0.00003$  for queued time, author experience, and size-related features, respectively.

**Model re-evaluation.** Our Observation 4(6) shows that the build history features have a strong explanatory power of our model. Hence, we build naïve baseline models for the three features of the build history family, and compare the [AUROC](#) of these baseline models with our initial model discussed in Section 4.3.2. Firstly, regarding **the naïve baseline for the recent build status**, it posits that the current build status will mirror the previous one. A similar baseline approach had been discussed in another recent study as well [159]. In our case, this approach yields an [AUROC](#) of 0.7622. Secondly, **the naïve baseline for the timeout ratio** predicts a current build will timeout if the project’s historical timeout ratio surpasses 0.5, achieving an [AUROC](#) of 0.7212. Lastly, our **naïve baseline for recent build duration** assumes a current build will timeout if the duration of its predecessor exceeds CircleCI’s standard time limit (ten minutes).<sup>20</sup> This method results in an [AUROC](#) of 0.5, indicating a prediction no better than random chance. This baseline analysis highlights that our model’s performance ([AUROC](#) of 0.939) surpasses the above baselines.

---

<sup>28</sup><https://github.com/BranchMetrics/android-branch-deep-linking-attribution>

<sup>29</sup><https://www.json.org/json-en.html>

Timeout builds are strongly associated with the project build history and timeout tendency; however, queued time, author experience, and size are weakly associated with timeout builds.

### 4.3.4 Longitudinal Analysis

The interpretation of our model in Section 4.3.3 indicates that the project build history is a strong indicator of whether a build will time out or not. To better understand this chronological relationship between timeouts, we perform a longitudinal analysis of the occurrences of timeouts.

#### Approach.

To structure our longitudinal analysis, we draw inspiration from recent studies on build breakages [138, 85], which report substantial improvements by distinguishing between consecutive breakages—where the immediately preceding build also failed—and novel breakages—where the preceding build was successful. Similarly, we classify timeout builds as either isolated or clustered. Fig. 4.5 exemplifies each form. In the figure, circles denoted with an  $S$  represent signal-generating builds (either passing or failing), while circles denoted with a  $T$  represent timeout builds. The first three timeout builds form *clustered timeout builds*. The last two timeout builds in the timeline are *isolated timeout builds*; they occur in between two signal-generating builds.

To analyze clustered timeout builds, we compute the number of timeout builds that compose a cluster as well as the time duration of the builds in the cluster. For example, the cluster annotated in Fig. 4.5 is composed of  $T_1$ ,  $T_2$ , and  $T_3$ ; hence, the number of timeout builds in the cluster is three. This cluster’s time duration is the time between the moment

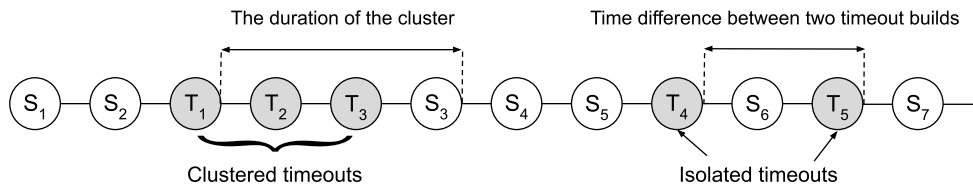
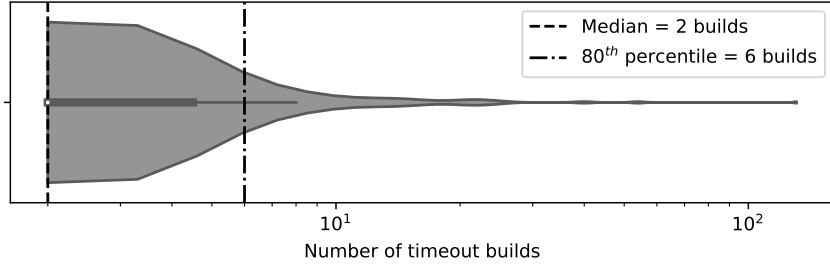
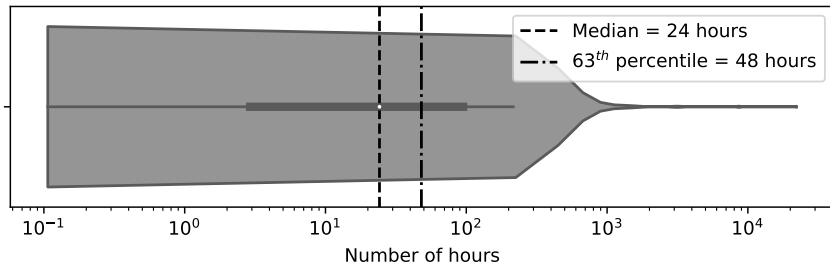


Figure 4.5: An example of builds timeline.



(a) Number of builds in timeout clusters.



(b) Duration of builds in timeout clusters.

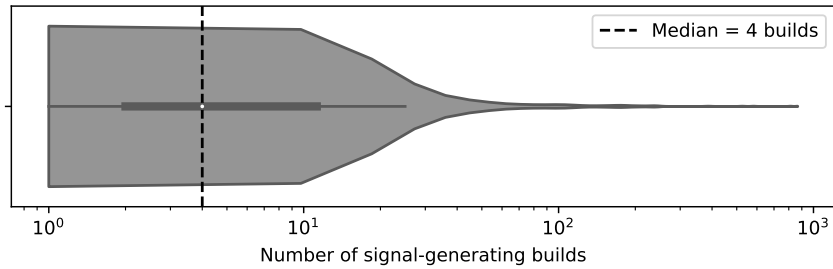
Figure 4.6: The number and duration of a timeout cluster.

that the first timeout in the cluster is observed and the moment the next signal-generating build is observed, i.e., the time between the end of  $T_1$  and the end of  $S_3$ .

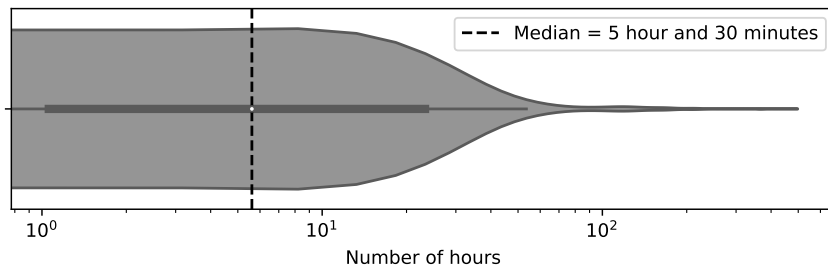
To analyze isolated timeout builds, we examine how close an isolated timeout build is to another timeout build (either a cluster or an isolated one), i.e., given an isolated timeout, we want to analyze the distance to the closest timeout to this isolated timeout. We measure the distance using the number of signal-generating builds between the isolated timeout and the closest timeout. Also, we analyze the time elapsed between an isolated timeout build and its closest timeout build. For example, consider the isolated timeout  $T_4$  in Fig. 4.5. The closest timeout build to  $T_4$  is  $T_5$ , which is also an isolated timeout.  $T_5$  is one build away from  $T_4$ ; hence, the number of builds between the two timeouts is one. The time duration until  $T_5$  has occurred is measured by the time difference between the ends of  $T_4$  and  $T_5$ .

## Results

Our two main observations of this RQ are detailed as follows.



(a) Distance (number of builds) between an isolated timeout and the closest timeout.



(b) Time difference (hours) between an isolated timeout and the closest timeout.

Figure 4.7: The distance and time difference between an isolated timeout and the closest timeout.

**Observation 4(9): The majority (64.03%) of timeout builds in our dataset occur in clusters.** Fig. 4.6a shows the distribution of the number of builds that compose a timeout cluster. The figure shows that although timeout clusters are composed of a median of two builds, 20% of the clusters are composed of at least six consecutive timeouts (see the 80<sup>th</sup> percentile). Fig. 4.6b shows the distribution of the duration of timeout clusters. Accordingly, it takes a median of 24 hours before a signal-generating build occurs. More extremely, we find that it takes at least 48 hours in 37% of the timeout clusters (see the 63<sup>th</sup> percentile). For example, the `cptactionhank/docker-atlassian-confluence` project<sup>30</sup> encountered a cluster of 14 consecutive timeout builds, and it took more than six days for the timeouts to subside.

**Observation 4(10): Isolated timeout builds are often close to another timeout build.** Fig. 4.7a shows the distribution of the number of builds between an isolated

<sup>30</sup><https://github.com/cptactionhank/docker-atlassian-confluence>

timeout and its closest timeout, and accordingly, the median is four builds. Furthermore, Fig. 4.7b shows the time difference between two timeouts, one of which is an isolated timeout. Indeed, the closest timeout to an isolated timeout build occurs in less than a day (i.e., five hours and 30 minutes) on the median. Examples of such isolated timeout builds can be found in the `spotify/helios` project.<sup>31</sup> This project had encountered timeout builds in less than an hour (in the extreme case) after an isolated timeout build. Similar cases can be observed in other projects, such as `cptactionhank/docker-atlassian-confluence`,<sup>30</sup> `palantir/atlasdb`,<sup>21</sup> and `onyx-platform/onyx`.<sup>32</sup>

The majority (64.03%) of timeout builds occur in clusters. Moreover, after a timeout build, it takes a substantial amount of time (a median of 24 hours) until the occurrence of a signal-generating build.

### 4.3.5 Thematic Analysis

The discovery of timeout clusters in Section 4.3.4 and the key influential features in Section 4.3.3 led us to explore the root causes behind CI timeouts, and below, we detail our approach and results.

## Approach

We start by gathering links to community discussions on timeout builds in the projects that we analyzed in our study by using GitHub search, with queries like “ci timeout” and “circleci time out” to find relevant issues and PRs. By analyzing these, we aim to understand the reasons behind CI timeouts. Our search finds 79 issues and PRs with 406 comments related to CI timeouts in the projects examined. After collecting relevant documents, we perform a systematic inspection and a thematic analysis [144]. In the initial iteration, two coders collaboratively review titles, descriptions, and discussion threads (i.e., comments) to create codes summarizing reasons for timeout builds. If the reason remains unclear, the coders mark it as “Unknown.” The coders then identify common themes that span across codes (which are not necessarily mutually exclusive), linking together similar topics or underlying issues.

---

<sup>31</sup><https://github.com/spotify/helios>

<sup>32</sup><https://github.com/onyx-platform/onyx>

In the second iteration of the analysis, the two coders independently assign themes to discussions, with any disagreements resolved through discussion or, if needed, a deciding vote by a third coder. However, all disagreements were resolved without needing this vote. To assess the reliability of these themes and coding, we calculate Cohen’s Kappa coefficient [28]. This coefficient is used to evaluate inter-rater agreement for categorical items between the two coders. The coefficient ranges from -1 to +1, with values greater than zero indicating a level of agreement that surpasses what would be expected by chance alone. In our case, the Cohen’s Kappa coefficient is 0.778, showing substantial agreement [97].

## Results

Our thematic analysis yields six themes, as shown in Table 4.5. Each theme describes a reason for build timeouts and the solutions that developers implemented. The total frequency of the identified reasons is greater than 100% because multiple themes may apply to the same case.

**(T1) Efficiency Issues in Testing.** In this theme, testing issues cause timeouts for two reasons: First, extensive tests lead to timeouts, and prompt developers to exclude them from CI pipelines. For example, in the `cptactionhank/dockeratlassian-jira` project, developers have removed a set of long-duration tests to mitigate timeouts.<sup>33</sup> Second, timeouts may be caused by misconfigurations or an excessive number of tasks, which can prolong build times. For example, the CI pipeline in the `tikv/tikv` project is configured with a code coverage tool that prolongs the build time, and is to blame for timeouts.<sup>34</sup> The developers initiated executing this code coverage analysis in a separate build to reduce the prolonged build durations.

**(T2) Project-Specific Issues.** Timeouts arise from project-level parameters, such as the selection of programming languages, databases, and Android emulators. For example, in the `BranchMetrics/android-branch-deep-linking-attribution` project, the selected Android emulator becomes unresponsive, entering into infinite loops during the build process.<sup>35</sup> This unresponsiveness contributes to timeouts. To avoid such timeouts, the developers changed the version of the Android emulator they were using.

**(T3) Network Issues.** Another common reason for timeout builds is due to network issues, such as incorrect network settings, API network errors, and server timeouts.

---

<sup>33</sup><https://github.com/cptactionhank/docker-atlassian-jira/commit/dbb3b143efe02351614e6f33be4b02-39991f40f2>

<sup>34</sup><https://github.com/tikv/tikv/issues/3012>

<sup>35</sup><https://github.com/BranchMetrics/android-branch-deep-linking-attribution/pull/400>

Table 4.5: The extracted themes for CI timeout builds.

ID	Theme	Frequency (%)	Solution
(T1)	Efficiency Issues in Testing	28 (35.44%)	Remove long-running tests and/or execute long-running tasks in separate builds.
(T2)	Project-Specific Issues	10 (12.65%)	Project-specific patterns.
(T3)	Network Issues	8 (10.12%)	Increase the CI timeout setting.
(T4)	Resource Constraints	8 (10.12%)	Induce waiting in threads and/or processes.
(T5)	Efficiency Issues in the CI Provider	6 (07.59%)	Restart the build.
(T6)	Containerized Environment Issues	6 (07.59%)	Increase the CI timeout setting.
(TU)	Unknown	16 (20.25%)	Increase the CI timeout setting.

For instance, in the spacetelescope/notebooks project, slow network requests led to CI timeouts.<sup>36</sup> Attempts to address this included extending the timeout limit from 10 to 20 minutes, which, while reducing timeouts, is not ideal as it can increase costs due to longer billable service usage.

**(T4) Resource Constraints.** Timeout builds can be attributed to resource constraints, such as limitations in [Random Access Memory \(RAM\)](#) and/or [Central Processing Unit \(CPU\)](#), or parallelism constraints that manifest as race conditions. For example, in the tendermint/tendermint project, a statement to cause a thread to wait (`time.Sleep(time.Second)`) was added to mitigate race conditions that otherwise lead to timeouts.<sup>37</sup>

**(T5) Efficiency Issues in the CI Provider.** Builds may time out due to inefficiencies in the CI provider’s infrastructure. For example, in the influxdata/kapacitor project, developers observed that certain builds running quickly on local machines faced delays and timeouts on CircleCI.<sup>38</sup> Similarly, in a [PR](#) within the influxdata/influxdb project, developers noted timeouts, which were suspected to be due to the limitations of the infrastructure

<sup>36</sup><https://github.com/spacetelescope/notebooks/issues/87>

<sup>37</sup><https://github.com/tendermint/tendermint/issues/846>

<sup>38</sup><https://github.com/influxdata/kapacitor/pull/1631>

on the CI provider’s side.<sup>39</sup> In both cases, developers were in favour of restarting the build even though restarting without addressing the underlying issues can waste resources [109].

**(T6) Containerized Environment Challenges.** Timeouts within a containerized environment can be traced to container maintenance and configuration issues. For example, in the `moby/libnetwork` project, timeouts occurred because the network interfaces of Docker containers were not adequately cleaned up after the tests were completed.<sup>40</sup> On the other hand, in the `Homebrew/brew` project, Docker containers play a crucial role in the CI process by providing isolated environments for the building software packages.<sup>41</sup> This process times out when building large software packages, and the developers discussed the need to lift the timeout limit.

Emergent themes of root causes for CI timeouts range from technical challenges (T3, T4, and T6) and testing inefficiencies (T1) to project-specific (T2) issues and limitations with CI providers (T5).

## 4.4 Threats to Validity

In this section, we describe the threats to the validity of this chapter. To support verifiability and replicability, we make our replication package publicly available.<sup>22</sup>

### 4.4.1 Construct Validity

We have not measured all potential characteristics that impact timeout builds. For instance, the performance characteristics of the CI server, such as parallelism and scalability, are not included in our models. Such features may better explain the likelihood of timeout builds than the features we use. However, such information is not available publicly. To mitigate this, we select a set of 19 features spanning five dimensions of CI builds by consulting the related literature on the build outcome prediction [138, 25, 125] and defect prediction [161].

---

<sup>39</sup><https://github.com/influxdata/influxdb/pull/8961>

<sup>40</sup><https://github.com/moby/libnetwork/pull/1325>

<sup>41</sup><https://github.com/Homebrew/brew/issues/10597>

## 4.4.2 Internal Validity

We may have missed confounding factors that could impact our interpretations. For example, we observe that the longer the duration of the previous build, the higher the likelihood of CI builds timing out, but this might reflect limited CI service resources, making the observed relationship coincidental. Additionally, implicit factors of builds could be overlooked, such as the CI provider’s workload metrics (not available for us via public APIs/datasets), which might provide further context to our model.

Also, the time limit (the `no_output_timeout` setting), which developers can set, may have a relationship to the probability of timeout builds; when a project is assigned a larger time limit, the likelihood of encountering build timeouts naturally decreases. We collect `no_output_timeout` values for each build in our dataset (assuming the default time limit for the build that does not have the configuration explicitly set). Upon rerunning our models, we discover that the `no_output_timeout` settings do not significantly explain the model’s outcomes. For a more detailed exploration of this direction, we direct interested readers to our Online Appendix G,<sup>22</sup> which contains an in-depth overview of this updated model. Note that the goal of our study is to identify features that provide insights into the project’s overall health in terms of timeouts, irrespective of whether those features share causal or correlational relationships with timeout builds. We encourage future research to explore causal links between our features and timeout builds.

Our decision to use statistical models was made because of their ability to elucidate the influences of the set of studied features on timeout builds. We recognize that this choice might introduce bias, especially when compared to visually intuitive and interpretable machine-learning models, such as decision trees. For further analysis, we construct a decision tree using the same dataset. This decision tree does not yield substantially new insights, suggesting that this experimental design choice is not a substantial threat to the validity of our results. For a detailed exploration of our decision tree analysis, we invite readers to consult our Online Appendix H.<sup>22</sup> Moreover, we recognize that stronger causal claims would require access to server-side operational data, such as CI provider workload metrics, which are not available to us. Future researchers with access to such operational data may consider applying causal analysis techniques to further investigate the underlying drivers of CI timeouts.

Our thematic analysis shows that 20.25% of the issues we analyzed do not contain any discussion of a known root cause. In these cases, developers often increase the CI timeout setting as a mitigation strategy. Although the true root cause is not explicitly discussed in the issue comments, it is possible that developers are aware of it, but choose not to document it. As a result, the true frequencies for the root causes T1–T6 may be slightly

higher than what we report, and there may exist additional root causes beyond those we identified. However, we believe that this is not a substantial threat to the validity of our results, as we analyze a sufficiently large and representative sample of issues, and our coding process is composed of multiple rounds of review with high inter-rater agreement.

### 4.4.3 External Validity

Our models are built using data from projects that used CircleCI. As such, our results may not generalize to other CI services; however, there is nothing inherently service-specific about the phenomenon of timeout builds. Nonetheless, since we select statically-computable and CI service-agnostic features, our replication package<sup>22</sup> can be used to accelerate replication studies for other CI services (e.g., GitHub Actions, which is also known to be a popular CI service [63]). We select the 24 most timeout-prone projects for our analysis. As such, our results may not be generalized to all CircleCI users. We apply a set of conservative filters to exclude early-stage or immature projects where timeout builds are less relevant.

## 4.5 Practical Implications

In this chapter, we investigate the features that can characterize CI timeout by analyzing a dataset of 105,663 CI builds that span 24 open-source GitHub projects. Then, we conduct a thematic analysis to identify the root causes of CI timeouts by analyzing 79 issues and PRs with 406 comments related to CI timeouts in the projects examined. Below, we discuss the implications of our findings.

**Project build history and timeout clusters can provide useful information to proactively allocate resources (for CI providers) and minimize CI waste (for CI consumers).** Observation 4(6) shows that the history of a project’s builds is the strongest indicator of whether a build will time out. Additionally, we found that timeout builds occur in clusters—as discussed in Observation 4(9). By leveraging these project tendencies and timeout patterns, CI providers can anticipate timeout builds and take appropriate action. For example, if additional resources are available, it may be more cost-effective to proactively allocate them to builds with a high likelihood of timing out in order to mitigate such issues. This is by no means a simple action since one cannot know in advance the quantities of additional resources required to allow the problematic build to terminate with a pass or fail signal [56]. However, it is likely that timeouts will be retried [40], which will

likely cascade into a series of timeout builds (i.e., clusters), generating more waste than a proactive increment to the resources of the initial build would. After a timeout build, 24 hours are taken (on median) for a project to see a passing or failing build. Moreover, clusters of timeouts suggest a substantial problem, like a shared environmental condition or code change that introduces timeouts, rather than just flakiness. We recommend developers investigate the root causes (detailed in our Section 4.3.5) to better understand and prevent future timeouts.

**Prioritizing files that are prone to build timeouts can optimize resource allocation and may help to avoid incidences of timeout builds.** Observation 4(7) indicates that certain file characteristics can increase the likelihood of a CI build timing out. By inspecting examples of timeout builds, we find that certain files are more often implicated in timeouts. This may indicate that timeout builds are localized, and are often triggered by the changes to a small fraction of project files. Upon a closer inspection of our dataset, we find cases where developers make changes to certain files to fix the issue of timeouts, e.g., commenting out long-running test cases in test files,<sup>42</sup> adjusting test settings (such as changing the Android emulator version<sup>43</sup>), and changing CI configurations.<sup>44</sup> Hence, a tool that ranks such timeout-prone files based on the strength of their association with builds that time out may have a potential impact. For example, a tool may flag a change to a file if it is likely to eventually lead to a timeout build, letting project maintainers direct their efforts to optimize components that pose the most elevated risk for timeout builds. This strategy could prioritize the files that are frequently associated with timeouts. Specifically, files that have a consistent track record of leading to timeouts across multiple builds ought to be placed at the top of the prioritization list for analysis.

**Researchers should propose approaches for predicting timeout builds to assist developers in preventing timeouts.** Observations 4(3) and 4(4) show that our model achieves a high level of discriminatory power (AUROC = 0.939), is well calibrated in terms of risk estimates (Brier score of 0.008), and is highly stable for explaining the incidences of timeouts. However, our primary goal in this chapter is to use statistical models to characterize and understand timeout builds rather than to predict future timeouts. We encourage researchers to build upon our findings, i.e., the important features we identified in observations 4(6) and 4(7) and the timeout clusters in observations 4(9) and 4(10), as well as those in the context of build failure prediction, to develop more powerful prediction models for timeout builds. For example, prior work (e.g., [25]) built machine-learning

---

<sup>42</sup><https://github.com/cptactionhank/docker-atlassian-confluence/commit/e81db60ee6cbee71bb427aa0-15afb3b9762d029c>

<sup>43</sup><https://github.com/BranchMetrics/android-branch-deep-linking-attribution/pull/400>

<sup>44</sup><https://github.com/autoreject/autoreject/pull/194/commits/92b388594d3c1cb2678c1f189940b84cfc>

models to predict build failures, but our findings suggest that such approaches could be extended to predict timeout builds.

## 4.6 Chapter Summary

Compute resources that enable CI are a shared commodity that organizations need to manage. To prevent erroneous builds from consuming a large amount of resources, CI providers often impose a time limit. CI builds that exceed the time limit are automatically terminated. While imposing a time limit helps to prevent abuse of the service, builds that time out consume the maximum amount of resources and leave project team members without an indication of whether their change set will pass the CI build or not. Therefore, understanding timeout builds and the features that characterize them is important for improving the stability and quality of a CI provider. In this chapter, we investigate the prevalence of timeout builds and the characteristics associated with them. The highlights of this chapter are as follows:

- Project teams and CI providers can leverage a project’s build history to predict CI timeouts.
- Certain change sets associated with specific files in a project may have a higher likelihood of triggering CI timeouts. Instead of merely restarting a CI build after a timeout, project teams can prioritize investigating these high-risk files to identify and address the root causes.
- Project teams may also use the curated catalog of timeout root causes from our study to guide their debugging efforts.

**Concluding Remarks.** Although this chapter focused on inefficiencies in CI environments (i.e., CI timeouts), inefficiencies in project dependencies can also contribute to the waste of CI resources. In the next chapter, we set out to study inefficiencies in dependencies that can tacitly waste CI resources.

# Chapter 5

## Unused-Dependency Updates

**Note.** An extensive version of this chapter appears in the proceeding of Foundations of Software Engineering [175].

### 5.1 Introduction

Modern software systems rely on ecosystems of reusable code [151, 29], typically in the form of packages (a.k.a. dependencies) provided by dependency managers like `npm` (for JavaScript), PyPI (for Python), and Maven (for Java). Such external packages facilitate cross-project reuse [12] and improve developer productivity [89].

Dependencies in projects can introduce compatibility issues [188, 98, 16] and security vulnerabilities [188, 6, 7]. To address these issues, dependency developers frequently release new versions to fix bugs [27, 188, 98], enhance security [127, 128], and add new features [185, 95]. To benefit from the latest dependency versions, project teams actively maintain their dependency specifications. As projects evolve, managing dependencies can become increasingly complex, which in turn can lead to the accrual of unused dependencies [98, 152]. These unused dependencies can account for up to 59% of a project's declared dependencies [98]. They bloat the dependency folder [153] and introduce security vulnerabilities [15]. Such dependencies also prolong CI build times by adding unnecessary download and installation steps, as noted by developers.<sup>45</sup> While caching dependencies

---

<sup>45</sup>[https://github.com/E3SM-Project/e3sm\\_diags/pull/385](https://github.com/E3SM-Project/e3sm_diags/pull/385)

mitigates the waste generated by such cases [55], CI builds triggered from updates to unused dependency versions, i.e., *unused-dependency commits*, are entirely wasted, as they do not affect the actual build outcomes. In this chapter, we aim to quantify and characterize CI waste from builds triggered by updates to unused dependencies. More specifically, we analyze a dataset of JavaScript projects that use `npm` dependencies and `GHA`,<sup>8</sup> and answer the RQs below.

**(RQ1) What is the prevalence of CI waste due to unused dependencies?** Understanding the extent to which CI resources are wasted due to updates to unused dependencies is crucial for CI consumers and providers. For CI consumers, such as developers and project maintainers, identifying substantial sources of waste can highlight the need for more efficient resource allocation or alternative optimization strategies. For CI service providers, recognizing this waste may present opportunities for improving operational efficiency, whether through enhanced resource allocation or the adoption of new optimization strategies. In this RQ, we quantify CI waste from the perspectives of CI consumers as well as CI providers.

**Results.** Our findings reveal that unused dependencies are a substantial source of waste in CI builds. From the perspective of the CI provider, we find that 55.88% (3,427 build hours) of the overall CI build time that is consumed by updates to `npm` dependency specifications in the studied projects is attributed to unused dependencies. At the project level, a median of 56.09% of CI build time is spent on updates to unused dependencies. To provide an operational cost perspective on this quantity of CI waste, we compare the waste of the most wasteful projects with the monthly budget of free build minutes that is provided to projects by `GHA`.<sup>46</sup> Among those projects, we find that the CI build time that is spent on unused-dependency updates in 14 of the 54 studied months already exceeds their monthly allocation of free build minutes.

**(RQ2) What are the main sources of CI waste due to unused dependencies?** Since developers as well as bots (such as `Dependabot`<sup>4</sup>) update versions of dependencies in projects, to tailor effective waste reduction strategies, we must better understand *who* (i.e., either developers or bots) are generating the majority of unused-dependency commits. For example, if bots emerge as the primary contributors of unused-dependency commits, an effective waste mitigation strategy would need to include scrutinizing and refining the functionalities and configurations of these bots. Conversely, if developers are identified

---

<sup>46</sup><https://docs.github.com/en/billing/managing-billing-for-github-actions/about-billing-for-github-actions>

as the primary contributors of unused-dependency commits, an effective waste mitigation strategy would need to raise awareness about the importance of dependency management. Besides, it is equally important to understand *which* types of dependencies tend to be affected (i.e., development or runtime). If the majority of the unused-dependency updates are made on development dependencies, mitigation strategies should focus on those dependencies and vice versa.

**Results.** Our results reveal that a large proportion (92.93%) of the CI build time that is spent on unused dependencies is wasted due to bot-generated updates, with **Dependabot** accounting for 74.52% of that wasted CI build time. With respect to the type of dependencies, the majority of the wasted CI build time (92.63%) occurs due to unused development dependencies, which are at lower risk of introducing field failures due to erroneous removal [35]. This suggests that development teams who are willing to invest in removing unused dependencies can focus on these development dependencies to reduce CI waste without exposing projects to elevated risk levels.

Our results show that a substantial quantity of CI waste is generated by unused-dependency commits. Existing CI service providers do not have measures in place to minimize this waste. Thus, we develop an approach to mitigate CI waste by skipping builds when they are triggered by unused-dependency commits, i.e., **DEP-SCIMITAR**. A retrospective analysis of the application of **DEP-SCIMITAR** to past commits in the studied projects shows that 68.34% of wasted CI build time can be saved with a precision of 94%.

## 5.2 Study Design

In this section, we describe our study design. Specifically, we present our approaches to project selection (PS) and data curation (DC). Fig. 5.1 provides an overview of the steps involved in these approaches, which we describe below.

### 5.2.1 (PS) Project Selection

Our study aims to analyze the waste in the CI process that is generated by version updates to unused dependencies. Therefore, we need to collect a dataset of projects that have accrued a rich history of dependency changes and build logs, as well as transparent CI configurations. Below, we describe the steps that we follow to select our sample of projects for analysis.

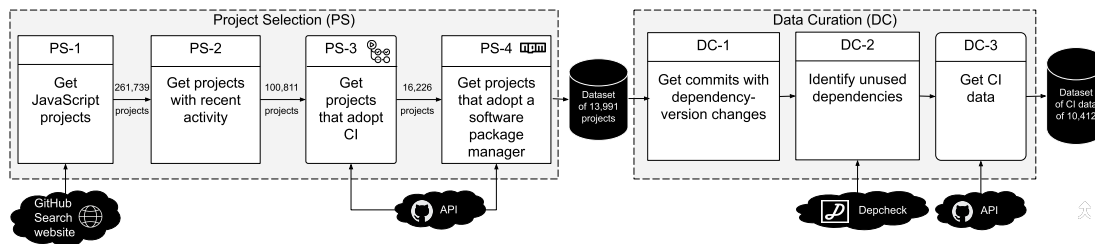


Figure 5.1: An overview of our study design.

**(PS-1) Select JavaScript projects.** We use the SEART GitHub search engine [31] to query for GitHub projects that meet our basic inclusion criteria. We select JavaScript projects due to JavaScript’s popularity and importance. In fact, JavaScript is currently the most popular programming language in the world, with a vibrant and fast-growing ecosystem.<sup>47,48</sup> This rich ecosystem is a boon for developers, and our query returns 261,739 JavaScript projects.

**(PS-2) Select projects with recent activity.** Since GitHub hosts toy and immature projects [118, 31, 90], we remove projects with fewer than ten commits. Inspired by prior work (e.g., [115]), we purposely do not restrict our project dataset only to the most active projects (i.e., projects with a large commit history) because there can be projects that are not updated frequently, but still play a critical role in the build process of other projects. Nonetheless, to ensure that the projects that we study have been active recently, we select projects that have received commits within the January 2020 to December 2022 timeframe. This filtering criterion improves the validity and modern relevance of the conclusions that we draw. After applying these filters, 100,811 projects survive.

**(PS-3) Select projects that adopt CI.** To analyze CI waste, we need to select projects that actively apply CI. To do so, we choose to select projects that adopt the **GHA** CI service,<sup>8</sup> which has quickly become the predominant CI service among **npm** projects on GitHub [63]. Furthermore, as of December 2022, **GHA** had accumulated a catalog of over 16,000 reusable actions [36]. To identify projects that are configured for **GHA**, we use the GitHub API<sup>49</sup> to check for the availability of the corresponding CI configuration files in each project. In particular, we check for the presence of `.yaml` files within the `.github/workflows` directories of the candidate projects.<sup>8</sup> We find that **GHA** is configured for 16,226 projects in our dataset.

<sup>47</sup><https://www.npmjs.com>

<sup>48</sup><https://libraries.io/NPM>

<sup>49</sup><https://docs.github.com/en/rest/repos/contents?apiVersion=2022-11-28>

**(PS-4) Select projects that adopt a software package manager.** `npm` is the de facto package manager used by JavaScript projects to manage their dependencies [172]. Therefore, we select projects that adopt `npm`. JavaScript projects that adopt `npm` must specify their dependencies in a `package.json` file, a JSON file that lists the packages upon which this project depends, as well as their versioning constraints. A project with `npm` dependencies must contain at least one `package.json` file located in its root folder. To identify projects that use `npm`, we first clone a local copy of each of the 16,226 candidate projects that have been selected so far. Then, we search the root directory of the HEAD commit of each cloned repository for a `package.json` file. If a match is found, we store the repository for further analysis.

At the end of this filtering process, 13,991 projects survive. These projects have a median of 181 commits and seven contributors. Our corpus of candidate projects comprises popular and large projects from organizations of influence, such as Meta,<sup>50</sup> Google,<sup>51</sup> and Microsoft.<sup>52</sup> We made this dataset is available online.<sup>53</sup>

## 5.2.2 (DC) Data Curation

After obtaining our set of projects, we process each project further to calculate the time that was spent on builds that were invoked due to updates to unused dependencies. An overview of the steps of this process is shown in Fig. 5.1 (DC), which we detail below.

**(DC-1) Extract commits with dependency-version updates.** We need to extract the dependency changes to identify updates to dependency versions. As explained above, JavaScript projects specify their direct dependencies in the `package.json` file, which contains the list of packages upon which the project depends. Hence, we extract all changes (i.e., commits) that modify the `package.json` file. Specifically, for each project, we mine through its commits, extracting the list of modified files, and the content of the modified lines using the `git-log` command. Note that to ensure the modern relevance of our analysis, we only select commits that occurred between 2020 and 2022 (inclusive). Then, we categorize these commits into those pertaining to dependency-version updates and those unrelated to dependency versions. Specifically, we consider a commit as a *dependency commit* if it exclusively modifies the `package.json` file (or both `package.json` file and

---

<sup>50</sup><https://github.com/facebookincubator/rapid>

<sup>51</sup><https://github.com/googlechromelabs/tooling.report>

<sup>52</sup><https://github.com/microsoft/react-native-macos>

<sup>53</sup><https://doi.org/10.5281/zenodo.11192753>

`package-lock.json` file) and the only updates in that file are version specifiers of dependencies. Commits that do not meet both criteria are considered outside the scope of our investigation and are not considered in our analysis of CI waste. We detect 121,453 dependency commits spanning 1,854 projects.

Note that our approach may lead to the exclusion of commits that update unused dependencies in the `package.json` file while simultaneously making changes to other files that do not impact the source code functionality. For instance, commits that merely add comments in source files or modify the `README.md` file ([3]), in addition to updating unused dependencies in the `package.json` file, fall outside the scope of our analysis. As a result, the waste that we report represents a conservative lower bound, underestimating the actual quantity of CI waste accrued due to unused-dependency commits.

**(DC-2) Identify unused dependencies.** Following prior studies [89, 83, 121], we apply `DEPCHECK`<sup>54</sup> to identify unused direct dependencies that are listed in the `package.json` file. First, we run `DEPCHECK` on each dependency commit and store a list of all unused dependencies. Subsequently, we cross-reference this list of unused dependencies with those that are modified in the commit to identify relevant commits. When a match occurs, we classify the commit as a dependency commit responsible for modifying the version of an unused dependency, a.k.a., an *unused-dependency commit*. Of the 121,453 dependency commits that we extract in DC-1, 49,731 are unused-dependency commits, which are of particular interest to us due to their potential to contribute to CI waste by triggering CI builds.

**(DC-3) Extract CI data.** To analyze the amount of CI waste that is generated by unused-dependency commits, we retrieve the CI data that is associated with them using the GitHub API.<sup>55</sup> Note that not all of these commits are directly associated with CI builds. A subset of them fail to trigger CI builds altogether, while in other cases, gathering CI data from the API is no longer possible, often because the data is no longer available. Consequently, we could retrieve CI data for only 20.9% of the unused-dependency commits.

We acknowledge that CI data is noisy [60, 57, 49]. This noise stems from various sources, such as experimental builds that fail, passing builds with ignored failing steps, and timeouts without proper signals. However, within the scope of our research, this noise does not pose a substantial concern because our study takes a holistic view of CI resources. In particular, we provide an estimation of CI waste from both the CI consumer and provider perspectives, which is a lower-bound estimate of the actual amount of CI waste. Indeed,

---

<sup>54</sup><https://github.com/depcheck/depcheck>

<sup>55</sup><https://docs.github.com/en/rest/checks/runs?apiVersion=2022-11-28>

since these noisy builds still consume CI resources, we consider them valid data entries for our study, and do not make attempts to filter them out.

Also, prior work [63] revealed that a number of GHA workflows are not entirely CI-related. For example, GitHub Actions are used for various purposes, such as manually triggering workflows (e.g., `workflow_dispatch`, accounting for 8.3%) and scheduled workflows (e.g., `schedule`, accounting for 8.1%). Our dataset does not contain such GitHub Actions that are not CI-related because we only collect the CI builds that are associated with dependency-update commits.

## 5.3 Study Results

In this section, we describe the approach to answer our RQs and the corresponding results.

### 5.3.1 (RQ1) What is the prevalence of CI waste due to unused dependencies?

Measuring the time spent on CI builds is crucial for better resource management. By analyzing the effect of unused-dependency commits on CI resources, we strive to provide insights for the following two primary stakeholders of CI:

**CI Consumers (i.e., project maintainers and developers).** Understanding whether a considerable amount of CI build time is spent on unused-dependency commits will help CI consumers direct their future efforts. For example, if a large amount of CI build time is spent on unused-dependency commits, it may impose a financial burden on project maintainers. While a quota of CI build time is provided for free on a monthly basis,<sup>46</sup> CI build time that is spent on unused-dependency commits wastes this limited resource and may push projects over the free limit into billable time. If only a small amount of CI build time is spent on unused-dependency commits, it may suggest that effort would be better spent on other resource-saving options (e.g., build-oriented refactoring [164]).

**CI Providers (e.g., GHA).** If the CI build time that is spent on unused-dependency commits is not billable, the CI provider must absorb the cost of that CI build time. Such wasted resources that are spent across a large number of projects will quickly accrue. Even if the wasted CI build time is billable and the cost is borne by consumers, these wasted resources still indicate inefficiencies in resource allocation and present an opportunity for the optimization of CI operations.

## Approach

Our quantification of CI waste resulting from version updates to unused dependencies offers insights that are tailored to the distinct perspectives of each CI stakeholder.

**CI Consumer.** In this perspective, we stratify our analysis based on individual projects. For each project in our dataset, we count the commits and compute the CI build time that is associated with builds that were triggered due to unused dependencies. This approach sheds light on the concern from the perspective of project maintainers, offering insights into how this problem impacts different projects within the open-source community. For example, it provides insights regarding the unnecessary maintenance activity that is generated by unused dependencies.

**CI Provider.** From this perspective, our investigation focuses on quantifying the combined influence of unused-dependency commits on CI build time. In other words, we count the commits and compute the CI build time that is associated with builds that were triggered due to unused dependencies. This viewpoint emphasizes the cost incurred by the CI provider.

## Results

Table 5.1 shows the proportion of commits and build hours that are generated by version updates to unused dependencies from the CI provider and consumer perspectives.

**Observation 5(1): More than half (55.88%) of the CI build time for dependencies is spent on CI builds that are triggered by unused-dependency commits.**

Table 5.1: The prevalence of CI waste from unused dependencies. The table presents the total and wasted number of commits and builds. The table further presents figures for both CI providers and consumers.

	CI Provider		CI Consumer	
	Commits	Build Hours	Commits (Median)	Build Minutes (Median)
Total Quantity (#)	20,743	6,133	7	42.13
Wasted Quantity (#)	10,412	3,427	3	23.63
Wasted Quantity (%)	50.19%	55.88%	42.85%	56.09%

This equates to a substantial waste of 3,427 build hours, originating from 50.19% (10,412 commits) of all dependency commits. A closer inspection reveals that 30% (3,123) of these unused-dependency commits consume more than ten minutes of CI build time for each commit, and 7% (728) consume more than half an hour for each commit. For CI providers, the fact that unused-dependency commits make up over half of all dependency commits presents an opportunity to reduce waste.

Dependency update commits account for a median of 2.30% of the total number of commits during the studied period, whereas unused-dependency commits account for a median of 1.09%. Across all projects in our dataset, the overall percentage of dependency commits is 3.21%, and the percentage of unused-dependency commits is 1.60%. Despite these modest percentages, the impact on CI build time is non-negligible and should not be ignored.

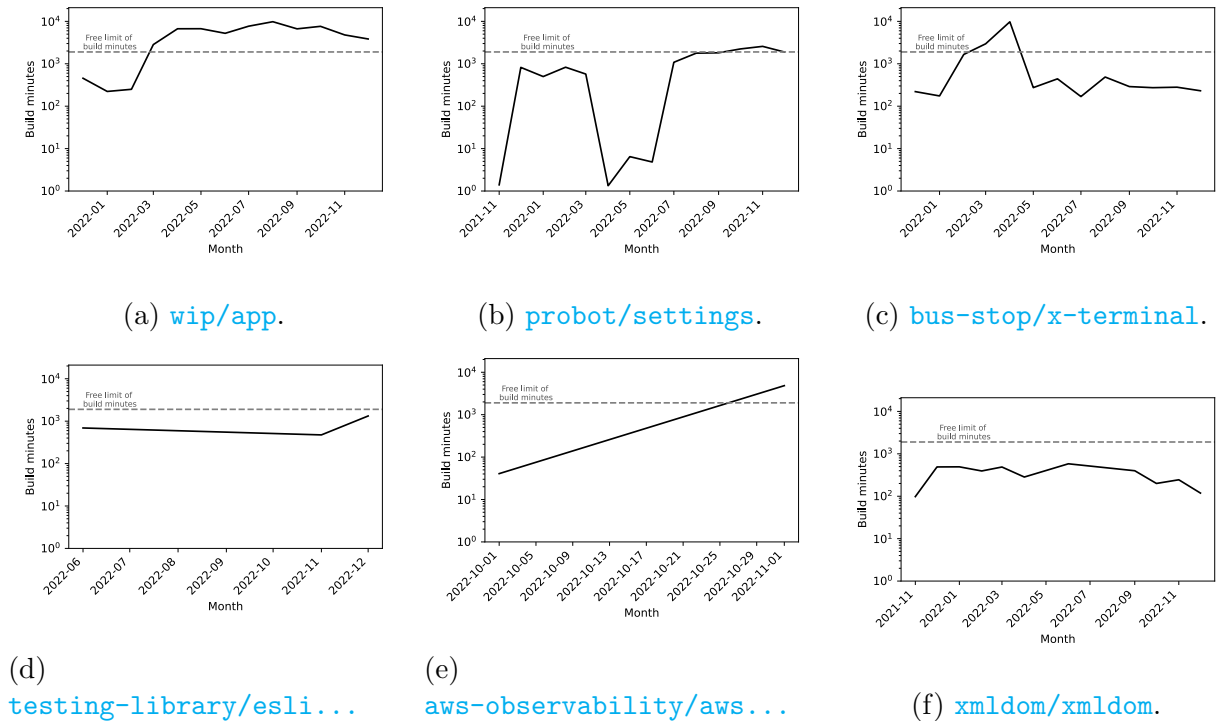


Figure 5.2: CI build time consumption of top six projects per month due to unused-dependency commits. The graphs corresponding to other projects are provided in our online appendix.<sup>53</sup>

**Observation 5(2):** At the project level, a median of 56.09% of the CI build time that is spent on dependency commits is generated by unused-dependency commits. The median CI build time that is consumed by unused-dependency commits in a project is 23.63 minutes. In fact, 30% of the projects in our dataset exceed an hour of wasted CI time. For example, the [dekkerglen/cubecobra](#) project consumed an hour of CI build time on unused-dependency commits, making up 31% of its total CI build time for all of its dependency commits. In more extreme cases, such as the [bus-stop/x-terminal](#) project, an alarming 127 build hours are wasted, with over 70% of its dependency-related CI build time being wasted on unused dependencies.

To provide an operational cost perspective on this quantity of CI waste, we compare the project-level waste with the budget of 2,000 build minutes per month that is provided to projects by the free plan of [GHA](#).<sup>46</sup> We conduct a focused analysis of the top six projects that accrue the largest amounts of CI waste. For this analysis, we use [GHA](#)'s billing criteria. In fact, [GHA](#) calculates build minute usage for billing based on factors, such as the platform that was used (Linux, Windows, or MacOS) for the execution of the build. Thus, for each build (triggered by unused-dependency commits), we identify the platform on which the build was executed from the CI data that we retrieve from the GitHub API during the DC-3 step in Section 5.2.2.

Fig. 5.2 shows the number of build minutes that are consumed by unused-dependency commits per month. The figure shows instances where this wasted CI build time alone already exceeds the entire monthly budget of free CI build time for these projects. Indeed, 14 of the 54 studied monthly periods exceed the free monthly budget. In the most extreme case, the [bus-stop/x-terminal](#) project wasted 9,756 build minutes in April 2022, exceeding the entire monthly budget of free build minutes by almost fivefold. In other months, the wasted CI build time of this project still constitutes a substantial portion, comprising at least 8.4% (168 build minutes) of the available free CI build time for the project.

Although both successful and failed builds triggered by updates to unused dependencies are considered wasteful due to their resource consumption, one may argue that failed builds are not actually wasteful because they usually raise concerns that developers should address. To address this viewpoint, we conduct a follow-up analysis to examine the percentage of successful and failing builds in our dataset. We find that 87.61% of the builds that are associated with unused-dependency updates are successful, while 12.39% failed. To explore the impact of considering only successful builds as wasteful, we conduct a revised prevalence analysis. This analysis reveals that wasted builds from unused-dependency commits account for 43.98% of the dependency-update commits in our dataset, with wasted CI build time comprising 38.19%.

Unused dependencies are a substantial source of inefficiency in CI processes. For CI providers, more than half (55.88%) of the CI build time of dependency-update commits is taken up by unused-dependency commits in the studied projects, generating a considerable amount of waste (3,427 build hours). At the project level, the median project spends 56.09% of its dependency-related CI build time on updates to unused dependencies. Among the six most wasteful projects, more than their entire monthly budget of 2,000 free build minutes is entirely spent on building unused-dependency commits in 14 of the 54 studied months.

### 5.3.2 (RQ2) What are the main sources of CI waste due to unused dependencies?

In Section 5.3.1, we observe that a considerable amount of CI waste is generated by unused-dependency commits. Understanding the origin of such commits is essential for crafting targeted solutions. In this section, we characterize CI waste according to the type of (1) commit author and (2) dependencies being updated.

**Commit Authorship.** Unused-dependency commits might be created by people maintaining the project or an automated software bot. Prior research [8, 78, 117] suggests that projects often use automated bots, such as Dependabot, to keep their dependencies up to date. If bots emerge as the primary contributors of unused-dependency commits, an effective waste mitigation strategy would need to include scrutinizing and refining the functionalities and configurations of these bots. Conversely, if developers are identified as the primary contributors of unused-dependency commits, an effective waste mitigation strategy would need to raise awareness about the importance of dependency management. The time that developers are spending on this unnecessary maintenance of unused dependencies could be better spent on more productive and impactful development activities.

**Dependency Type.** According to official guidelines,<sup>56</sup> npm dependencies may be *development* and *runtime*. Development dependencies are used during development and testing, and are listed in the `devDependencies` section of the dependency specification file (`package.json`). For instance, `webpack`<sup>57</sup> is a development dependency in JavaScript projects, bundling modules for delivery on the web. Runtime dependencies are necessary for production deployment environments. An example would be `lodash`,<sup>58</sup> which provides

---

<sup>56</sup><https://docs.npmjs.com/specifying-dependencies-and-devdependencies-in-a-package-json-file>

<sup>57</sup><https://webpack.js.org/concepts>

<sup>58</sup><https://lodash.com>

implementations of data structures like arrays and strings. We strive to understand the type of dependencies that generate most of the waste to formulate effective mitigation strategies. For example, if most unused-dependency commits update development dependencies, they would present less risk-prone opportunities for optimization, as they are used solely during the development phase and are not installed in production environments.

## Approach

Our approach to understanding the sources of unused-dependency commits focuses on (1) identifying who made the commit and (2) categorizing the type of dependency.

**Commit Authorship.** To distinguish between bot-generated and developer-generated waste, we identify who authored each unused-dependency commit. Following prior work [38], we apply a regular expression to detect authors having the term “bot” in their name, classifying them as bots. All other authors are labeled as developers. Our analysis reveals four bot candidates, and these are indeed automated bots: `Dependabot`,<sup>4</sup> `Renovate bot`,<sup>59</sup> `Snyk bot`,<sup>60</sup> and `Depfu bot`.<sup>61</sup> For the remaining authors, we estimate the accuracy by inspecting a sample of 400 randomly selected commits, which provides a 95% confidence level that our observed proportions are within a confidence interval of  $\pm 5\%$ . We inspect each sampled commit to determine if its author is a bot. If this inspection of the author’s name is inconclusive, we then cross-reference the name with the corresponding GitHub profile to arrive at a decision. However, this analysis yields no instances of incorrect labeling. To account for potential name variations or aliases, we employ heuristics to consolidate identities [169]. After establishing commit author categories, we compute both the number of unused-dependency commits and the total CI build time that is consumed by these commits.

**Dependency Type.** To understand how CI waste due to unused-dependency commits is associated with the different dependency types, we examine the nature of dependencies that cause CI waste. We group unused dependencies into development and runtime categories according to whether they appear in the `devDependencies` or `dependencies` section of the `package.json` file, respectively. We analyze each unused-dependency commit by extracting both development and runtime dependencies from the `package.json` file as of the commit’s timestamp. After obtaining the list of dependencies, we cross-reference

---

<sup>59</sup><https://www.mend.io/renovate>

<sup>60</sup><https://snyk.io>

<sup>61</sup><https://depfu.com>

Table 5.2: Distribution of unused-dependency commits and corresponding build hours over bots and developers.

	<b>Bot</b>	<b>Developer</b>
Commits	9,280 (89.12%)	1,132 (10.88%)
Build hours	3,184 (92.93%)	242 (07.07%)

Table 5.3: Build hours attributed to unused-dependency commits authored by bots.

<b>Bot</b>	<b>Commits</b>	<b>Build Hours</b>
Dependabot	6,541	2,373
Renovate bot	2,514	741
Snyk bot	174	60
Depfu bot	51	10
Total consumption	9,280	3,184

it with the unused dependencies that we identify for each commit in Section 5.2.2 (DC-2) to determine whether an unused dependency is a development or runtime dependency. Then, we calculate the CI waste for each dependency type by counting the number of wasteful builds and their associated wasted build hours.

## Results

Tables 5.2 and 5.3 present the number of commits and build hours that are associated with unused-dependency commits, respectively.

**Observation 5(3): Bots are the primary contributors to the wasted build hours.** Table 5.2 shows that bots are responsible for a substantial proportion of the CI waste being generated by updates to unused dependencies. Specifically, bot-generated commits account for 3,184 build hours (92.93%) spanning 9,280 unused-dependency commits (89.12%). In contrast, developers account for only 7.07% of this wasted CI build time. The 1,132 developer-generated unused-dependency commits are produced by 265 developers. Even though this percentage is much lower than that of bots, it is important to highlight that a small number of developers are responsible for a substantial proportion of the generated waste. For example, the two developers who produced the most unused-dependency commits contributed 48% of the wasted CI build time. This suggests that

Table 5.4: Comparison of the total number of commits and build hours stemming from unused-dependency commits between development and runtime dependencies across all the projects in our dataset.

	Unused Development Dependencies	Unused Runtime Dependencies
Commits	8,762 (84.15%)	1,650 (15.85%)
Build hours	3,174 (92.63%)	253 (07.37%)

misinformed developers can quickly accumulate waste due to unnecessary maintenance.

Table 5.3 breaks down how each of the four detected bots contributes to CI waste. **Dependabot** emerges as the top contributor, representing roughly three-quarters of the wasted build hours (74.52%). **Renovate bot** is next, accounting for 741 hours (23.27%). The remaining **Snyk bot** and **Depfu bot** contribute 60 hours (1.88%) and ten hours (0.31%), respectively. This distribution closely follows the quantiles of unused-dependency commits that are associated with each bot.

**Observation 5(4): Unused development dependencies lead to most of the wasted build hours.** In Table 5.4, we see a clear distinction between the impact of unused development and runtime dependencies. Unused development dependencies contribute 92.63% of the total wasted time, amounting to 3,174 build hours. In contrast, unused runtime dependencies contribute only 7.37% (253) of the build hours. This suggests that development teams that are inclined to allocate resources to the mitigation of CI waste that is generated by unused dependencies would benefit most from focusing on these development dependencies. By doing so, they can mitigate CI waste efficiently while maintaining a minimal risk level of field failures, since these development dependencies are unlikely to affect the production environments [35].

Upon analyzing individual projects, we find that version updates to unused development dependencies consume a median of roughly five minutes of CI build time, while unused runtime dependencies consume a median of roughly three minutes; however, the distribution is skewed. Indeed, 19% of projects consume more than an hour and 40 minutes of wasted CI build time due to unused development dependencies. Comparatively, only 4% of projects waste that much CI build time for runtime dependencies.

Furthermore, Table 5.5 shows that while a total of 942 development dependencies are associated with unused-dependency commits, only five of them collectively contribute to 53.21% of the overall CI build time for unused development dependencies, which highlights the critical role being played by specific development dependencies that are implicated in unused-dependency commits.

Table 5.5: Build hours resulting from unused-dependency commits in development dependencies.

Unused Development Dependency	Build Hours
@vercel/node	797
eslint	325
prettier	245
jest	209
mocha	112
Other development dependencies (937 dependencies)	1,485
Total consumption	3,174

Bots contribute the vast majority (92.93%) of the CI build time that is wasted on unused dependencies. Moreover, unused *development* dependencies represent 92.63% of the wasted CI build time that is associated with unused dependencies in the studied projects. Thus, we recommend that waste mitigation strategies (1) target the most wasteful bots (i.e., **Dependabot** and **Renovate bot**) for improvements and (2) focus waste reduction efforts on development rather than runtime dependencies.

### 5.3.3 Mitigation of CI Waste Due to Updates to Unused Dependencies

Our results show that a substantial quantity of CI waste is generated by unused-dependency commits. Existing CI service providers do not have measures in place to minimize this waste. Consumers of CI services might not realize that unused dependencies linger in their projects [98], and removing unused dependencies can be an onerous task. For example, a discussion on the [GoogleCloudPlatform/nodejs-docs-samples](https://github.com/GoogleCloudPlatform/nodejs-docs-samples) project suggests that refactoring source code to remove such dependencies requires substantial effort.<sup>62</sup> Moreover, Kula et al. [95] found that developers are hesitant to make dependency-related changes in general due to the substantial efforts needed to avoid introducing errors.

Given the hesitancy among developers to introduce changes into a stable codebase due to the perceived risks, opting to skip unnecessary builds when they are triggered

<sup>62</sup><https://github.com/GoogleCloudPlatform/nodejs-docs-samples/pull/3168>

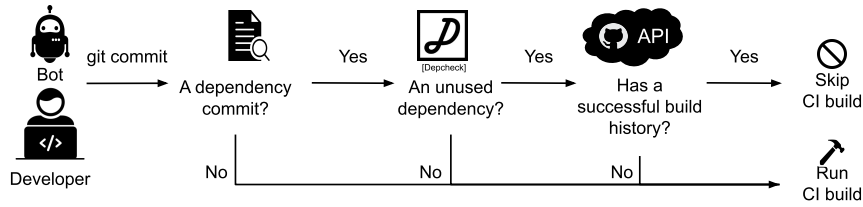


Figure 5.3: An overview of the DEP-SCIMITAR workflow.

by unused-dependency commits—which are skippable because they do not impact the project’s functionalities—emerges as a practical strategy. We perform an analysis of 272 GitHub issue reports related to removing unused dependencies (which is sufficient to provide a 95% confidence level with a 5% margin of error when making inferences about the entire population), and find that, in 40.44% of the cases, developers decide to remove the unused dependencies, while in 37.13% of the cases, developers decide not to remove unused dependencies; the other 22.43% of cases are not related to developer decisions. A more detailed description of this analysis is discussed in our Online Appendix.<sup>53</sup>

With these challenges in mind, we set out to develop an approach to mitigate CI waste by skipping builds when they are triggered by unused-dependency commits. This approach avoids the removal of such dependencies from the dependency specification files.

## Approach

We propose DEP-SCIMITAR—an approach to cut down on dependency-induced CI waste. Our approach is agnostic of the CI service provider, and is freely available as an `npm` package to foster its adoption.<sup>5</sup> Fig. 5.3 provides an overview of the design of DEP-SCIMITAR, emphasizing its automated reasoning process to skip CI builds for unused-dependency commits. Since unused-dependency commits are generated by humans as well as bots, DEP-SCIMITAR can process commits that are produced by both types of authors.

DEP-SCIMITAR begins when a commit generates a build request by analyzing the commit to check for updates to versions of dependencies in the `package.json` file. When such updates are detected, DEP-SCIMITAR uses the `DEPCHECK` tool to detect whether the changed dependency has an unused status. Although tempting, simply skipping such commits without considering their recent history may inadvertently permit a failing build to pass. For example, commit `#fa843b6` of the `aws-observability/aws-otel-js` project is an unused-dependency commit and it is linked to a failing build. A closer inspection reveals that its parent commit is also linked to a failing build, and the CI builds of both the current

and parent commits failed due to the same cause, i.e., a failure in the build step for testing a certain application feature. This implies that the error was carried forward without being fixed. To avoid such situations, whenever `DEP-SCIMITAR` detects an unused-dependency commit, it also retrieves the build status of the parent commit using GitHub API;<sup>55</sup> the tool skips the current build only if the build of the parent commit is successful. Note that our tool supports only public GitHub repositories due to its dependency on GitHub API calls for fetching build statuses without authentication tokens. For private projects, a project-specific authentication token is required.

## Configuration

The `DEP-SCIMITAR` prototype can seamlessly integrate into the CI process of consumers that adopt CI services, such as [GHA](#),<sup>8</sup> Travis CI,<sup>9</sup> and CircleCI.<sup>7</sup> Listing 1 shows a fragment from a [GHA](#) configuration file that uses `DEP-SCIMITAR`. The full version of this file is available in our Online Appendix.<sup>53</sup> This file needs to contain steps to both install our tool (`npm install dep-scimitar`) and run it (`npx dep-scimitar runremote`), as shown on lines 3 and 4 of Listing 1. Running the command classifies commits as unused-dependency commits or otherwise. This classification is stored as an environment variable (in line 5) and subsequently guides execution processes in the project, illustrated in line 8 (i.e., project-specific execution decisions).

```
1 - name: Check For Unused-Dependency Commits
2   run: |
3     npm install dep-scimitar # Install the tool
4     unusedDepCommit=$(npx dep-scimitar runremote) # Run the tool
5     echo "UnusedDepCommit=$unusedDepCommit" >> $GITHUB_ENV # Env
   variable
6
7 - name: Project-specific Steps
8   if: ${{ env.UnusedDepCommit == 0 }} # Check the env variable
9   run: |
10    # project-specific steps
```

Listing 5.1: Example of a configuration file required for `DEP-SCIMITAR`.

DEP-SCIMITAR<sup>🐛</sup> further offers an enhancement configuration to boost developer awareness. It can automatically add the `[CI skip]` tag to commit messages. This tag indicates to the CI service provider that the commit should not trigger a CI build. Commits with the `[CI skip]` tag skip builds on widely-used CI platforms, such as [GHA](#), Travis CI, and CircleCI. Users can leverage this feature by installing DEP-SCIMITAR<sup>🐛</sup> locally and enabling it for their project using the command: `npx dep-scimitar on`.

## Evaluation

To evaluate our approach, we create a prototype implementation that is compatible with `npm`-based GitHub projects. We conduct a retrospective analysis of past commits to the projects in our dataset, focusing on (1) the precision of the tool and (2) the reduction of wasted CI build time.

**Observation 5(5): The precision of Dep-sCImitar<sup>🐛</sup> is 94%.** While the precision is high, there are cases that this tool makes mistakes, falsely skipping builds that should not have been skipped. In particular, if the previous commit of an unused-dependency commit is linked to a passing build, DEP-SCIMITAR<sup>🐛</sup> skips the current commit because it updates an unused dependency and the commit does not have a failing history; yet, the current commit may fail if we do not skip it, and those cases are the false positives detected by our tool. A closer inspection reveals three scenarios that contribute to false positives, where the actual outcome of the build remains unknown. First, CI builds can fail due to external network issues, such as in the [mdn/bob](#) project, where a commit updating an unused dependency failed because it exceeded the API rate limit.<sup>63</sup> Second, CI build timeouts occur when the build process exceeds the allocated upper limit of build minutes, leading to automatic cancellation; for instance, a commit<sup>64</sup> in the [btargac/excel-parser-processor](#) project that updated an unused dependency caused the build to time out and be marked as a failure. Third, premature cancellations of CI builds will also result in a “failed” status. For example, a commit<sup>65</sup> in the [optimistiksas/oibus](#) project is associated with a failing build due to three canceled steps, despite its parent commit being linked to a successful build.

Note that calculating the recall is challenging. Below, we discuss two potential scenarios where DEP-SCIMITAR<sup>🐛</sup> may produce false negatives. First, we only choose to skip a build that is triggered by an unused-dependency commit if its preceding build succeeds. Upon

---

<sup>63</sup><https://github.com/mdn/bob/commit/013ff>

<sup>64</sup><https://github.com/btargac/excel-parser-processor/commit/5ba1>

<sup>65</sup><https://github.com/iobroker/iobroker.pushover/commit/905a>

closer inspection, we discover instances of unused-dependency commits where the previous build fails. Such preceding failures occur due to premature termination by the CI provider, e.g., when a higher-priority build is waiting in the queue. If the CI provider had not interrupted this build, it might have been successful. Such false-negative cases occur when our tool does not skip commits with a recent history of build failures and/or when commits include both an unused dependency version update and other minor changes, such as edits to documentation or comments in source files [3].

**Observation 5(6): Dep-sCImitar can save a substantial amount of the overall dependency-induced CI waste.** At the project level, we find that the median number of commits that are skippable is three, with a waste of 23 minutes; however, we find that 25% of the studied projects can save at least an hour and 23 minutes of CI build time. Projects like [probot/settings](#) skip 43 commits while saving 246.85 build hours. For a more detailed analysis of how such extreme projects generate waste in the context of the monthly budget of free build minutes that are provided by [GHA](#), in Fig. 5.4, we plot

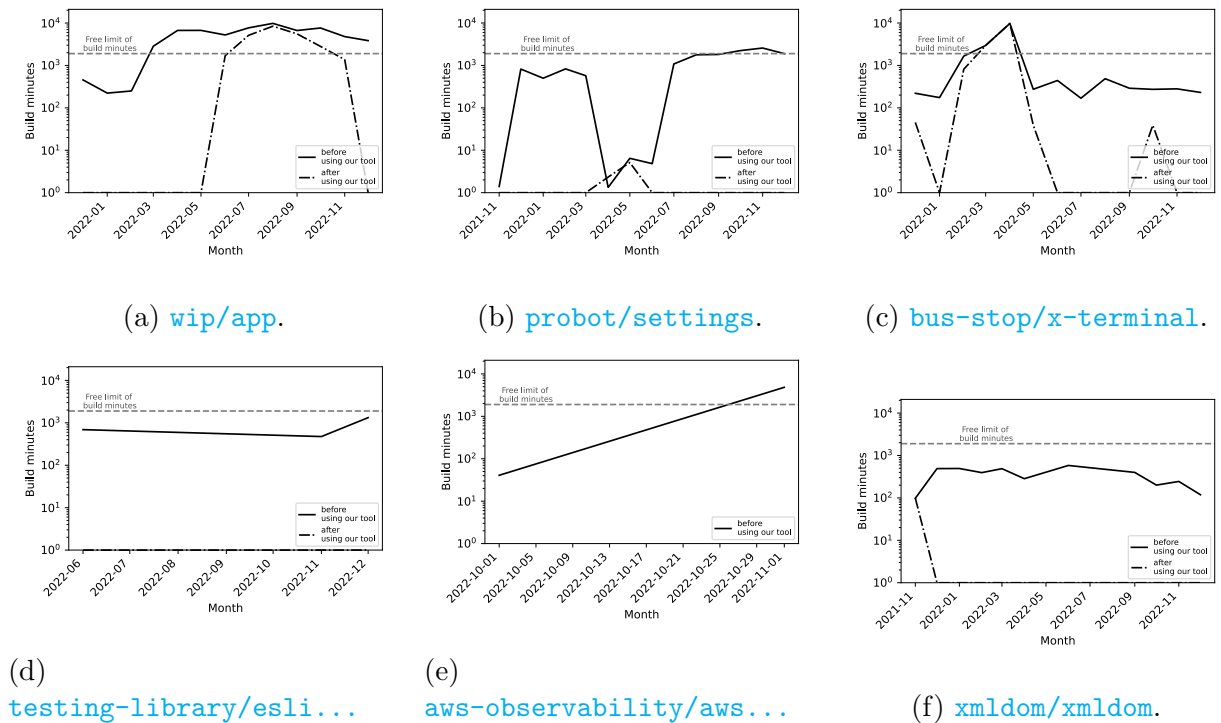


Figure 5.4: Distribution of the wasted CI build time of the top six projects before and after applying our tool.

the amount of wasted CI build time before and after applying DEP-SCIMITAR<sub>ℓ</sub> for the six projects that accrue the most dependency-induced CI waste. Focusing on these projects allows our analysis to effectively showcase the substantial savings and efficiency gains that are achievable with the DEP-SCIMITAR<sub>ℓ</sub> approach. This selection also acts as a proof of concept, indicating that similar benefits, if not proportionally scaled, could be achieved for other projects. For interested readers, the graphs that correspond to other projects are provided in our Online Appendix.<sup>53</sup> Time (in terms of months) is shown on the x-axis, and the number of build minutes that are spent is shown on the y-axis. The solid line shows the CI build time that was spent for all unused-dependency commits before applying the tool, and the dashed line shows the CI build time that would be spent after applying the tool. The gap between the two lines demonstrates the CI build time that is safely skippable by the tool.

Fig. 5.4 further shows that for the [probot/settings](#), [xmldom/xmldom](#), and [testing-library/eslint-plugin-jest-dom](#) projects, the CI waste is cut down to almost zero. For the [wip/app](#) and [bus-stop/x-terminal](#) projects, we observe that there are months when DEP-SCIMITAR<sub>ℓ</sub> cannot skip a substantial number of builds. This tends to occur when builds are linked to unused-dependency commits that were preceded by build failures. Nonetheless, we observe that a substantial amount of CI build time could still be saved for those projects in other months. Note that the [aws-observability/aws-otel-js](#) project has only two builds that are triggered by unused-dependency commits, and their parent commits are linked to failing builds, which DEP-SCIMITAR<sub>ℓ</sub> conservatively chooses to run. On the other hand, from the CI provider perspective, our tool detects 83% of wasteful CI builds, and could have saved 2,342 (68.34%) build hours.

Our DEP-SCIMITAR<sub>ℓ</sub> approach can effectively identify and skip CI builds that are associated with unused-dependency commits with a precision of 94%. A retrospective evaluation shows that integrating DEP-SCIMITAR<sub>ℓ</sub> in the studied projects would have saved 2,342 build hours, which amounts to 68.34% of the overall dependency-induced CI waste.

## 5.4 Threats to Validity

In this section, we describe the threats to the validity of this chapter. We made our dataset and scripts available online for replication purposes.<sup>53</sup>

### 5.4.1 Construct Validity

We use the `DEPCHECK`<sup>54</sup> tool to detect unused dependencies. Hence, we are limited by the accuracy of this tool. Validating unused dependencies can be challenging since `npm` packages can be loaded dynamically (e.g., using reflection mechanisms) that can go undetected by static analysis. While a dynamic tracing-based approach to identify dynamically loaded dependencies (e.g., `DEPRUNE` [104]) would improve the accuracy of detecting unused dependencies, such tools are known to identify a fewer number of unused dependencies than `DEPCHECK` [104] and require execution of the software, thus, are not lightweight solutions like static analyzers (e.g., `DEPCHECK`). Therefore, despite the potential for false positives, `DEPCHECK` remains, to the best of our knowledge, one of the most widely used tools for detecting unused JavaScript dependencies, and has been adopted in several prior studies [89, 83, 121]. Furthermore, the tool actively addresses false positives,<sup>66</sup> making it a practical choice for both developers and researchers.

### 5.4.2 Internal Validity

Inspired by prior work [38], we apply a regular expression to distinguish between commit authors who are bots and developers. Through this process, we identify four bots that are responsible for 9,280 unused-dependency commits. To mitigate false positives in our classification, we inspect the profiles and commit activity of these four bots and a sample of 400 commits; however, it is unlikely that our set of bots is complete. Thus, our bot-produced unused-dependency commit rates are best seen as minimum estimates rather than exact figures. In a similar vein, since GitHub allows the same commit author to use different names and/or different email addresses when committing changes,<sup>67</sup> there is likely noise in our authorship analyses (Section 5.3.2). To reduce the impact of aliases, we apply heuristics to consolidate them [169]; nonetheless, our findings should be viewed as estimates rather than precise figures.

### 5.4.3 External Validity

Our study is based on JavaScript projects that use `npm` and adopt `GHA` from 2020 to 2022 (inclusive). Hence, our findings might not directly apply to projects developed in other

---

<sup>66</sup><https://github.com/depcheck/depcheck/releases>

<sup>67</sup><https://docs.github.com/en/account-and-profile/setting-up-and-managing-your-personal-account-on-github/managing-email-preferences/setting-your-commit-email-address>

programming languages, those that use other CI providers, or commits made outside the specified timeframe. Also, our study’s reliance on the free build minutes quota as of August 2023 introduces the possibility that our findings may become outdated if there are changes to GitHub’s billing structure.<sup>46</sup> Despite these specifics, the key concept and the design of our study can still be applied to other settings to expand the investigation of inefficiencies in CI due to unused dependencies.

## 5.5 Practical Implications

In this chapter, we study dependency-induced CI waste. We collect and analyze a dataset of 20,743 dependency commits from 1,487 JavaScript projects that use `npm` dependencies. We find that 55.88% of the CI build time that is associated with dependency updates is wasted on unused dependencies. The median project allocates 56.09% of its dependency-related CI build time to updates of unused dependencies. Below, we distill the implications of this chapter for CI stakeholders, bot developers, and researchers.

**CI stakeholders (i.e., project maintainers and CI providers) should detect and omit unnecessary builds that are triggered by unused-dependency commits.** Observations 5(1) and 5(2) show that the projects that are most prone to CI waste spend a substantial amount of their quota of free build minutes on unused-dependency commits. This waste also contributes to financial costs for the CI provider, as well as increased development and maintenance costs. Our research calls for the attention of CI stakeholders to recognize which dependencies disproportionately generate CI waste and devise strategies to mitigate it. To put our results into action, we propose `DEP-SCIMITAR` to automatically recommend skippable builds to mitigate this waste without requiring a change to dependency configurations. A prototype implementation of `DEP-SCIMITAR` for JavaScript projects is publicly available.<sup>5</sup> Observations 5(5) and 5(6) show that `DEP-SCIMITAR` yields substantial benefits, detecting 83% of the wasteful commits, and avoiding 2,342 (68.34%) hours of CI build time that would have otherwise been needlessly expended.

**Bot developers should effectively manage CI waste due to unused-dependency commits.** Observation 5(3) shows that bots contribute the vast majority (92.93%) of the CI build time that is wasted on unused dependencies. Our findings shed light on the negative impact that bot-generated updates can have on project maintenance and their over-consumption of CI build resources. We recommend that bot developers incorporate tags, such as `[CI skip]`, into commit messages or [PRs](#) that update unused dependencies. Employing this tagging mechanism can enhance the recognition and screening of

such updates, allowing projects to allocate their CI resources to more pressing build requests. Additionally, we recommend that bot developers focus first on cutting down waste when they update development dependencies since these development dependencies are unlikely to affect production environments [35]. Observation 5(4) provides further support for this, indicating that a considerable amount of CI waste is linked to specific categories of development dependencies.

**Researchers should broaden the scope of the impact of unused dependencies beyond CI build time.** Although our study offers perspectives on the impact of unused dependencies on CI, future research should consider other implications that unused dependencies may have on development workflows and project maintenance. For example, our study reveals that unused dependencies are often maintained, e.g., by updating to more recent versions when they become available. This may be because organizations are sensitive to the security risks of using outdated dependencies [98], irrespective of whether the dependency is used. This practice also adds to development overhead, suggesting that exploring this area further could yield beneficial insights.

## 5.6 Chapter Summary

External packages, i.e., dependencies, are an essential part of a software project. Developers of these dependencies regularly release updated versions to provide new features, fix defects, and address security vulnerabilities. Thus, project teams who reuse these dependencies often try to keep their dependencies up to date to get the benefits of the latest improvements.

Due to the potential for regression, managing dependencies is not just a trivial matter of selecting the latest versions. As projects evolve, they tend to accrue dependencies, exacerbating the difficulty of dependency management. As a result, projects tend to accumulate unused dependencies. Although unused dependencies are not required to build and run the project, updates to their dependency versions can still trigger CI builds. These CI builds that are initiated by updates to unused dependencies are fundamentally wasteful. In this chapter, we study the CI waste that is generated by updates to unused dependencies. Below are the highlights of this chapter:

- CI builds that are triggered by updates to unused dependencies also waste a substantial amount of CI build time.
- Bots that trigger these wasteful builds need to be optimized to consider actual dependency usage before triggering updates.

- We introduce DEP-SCIMITAR<sup>5</sup>, a novel approach to safely skip CI builds triggered by unused-dependency updates.

**Concluding Remarks.** So far, we have focused on tacit inefficiencies in CI (timeouts and unused-dependency updates). To improve the effectiveness of CI, it is also important to mitigate tacit barriers, as they can impact collaboration in CI practices. To this end, in the next chapter, we set out to study tacit barriers in CI. In particular, we investigate whether there is a significantly greater lack of diversity among developers who contribute to CI artifacts in projects compared to those who contribute to other parts of the codebase.

**Part III**

**Tacit Barriers**

# Chapter 6

## Diversity of DevOps Contributors

**Note.** An extensive version of this chapter appears in the Empirical Software Engineering journal [177].

### 6.1 Introduction

While there is an inherent lack of diversity in the software engineering community, having diverse project teams has been associated with more resilient software solutions, and tends to improve the effectiveness of teamwork and problem-solving from broader perspectives [168]. In the context of CI, the diversity among team members who contribute to CI artifacts alongside other DevOps artifacts in a project (i.e., DevOps contributors) is a tacit barrier that can impact the overall performance of a project. In fact, a report, “Why diversity matters in DevOps,”<sup>1</sup> discusses the potential of diversity in DevOps teams to lead to more streamlined product development. Moreover, there is a growing focus on the diversity of DevOps contributors in the industry. For example, the global movement “Women in DevOps”<sup>68</sup> aims to close the gender gap in DevOps contributors, believing that a balanced and diverse workforce drives innovation.

DevOps contributors typically require expertise in collaboration, automation, measurement, monitoring [82], Agile methodologies [11, 10], and cloud-based tools and infrastructure [30, 10]. Such specialized skills and tooling expertise may not be as common in the wider open-source community. This is evident in the *Stack Overflow Survey (2022)*,<sup>17</sup> as

---

<sup>68</sup><https://www.womenindevops.com>

it shows that only 10.06% of respondents identified themselves as DevOps contributors. Note that this percentage is as low as 1.7% in *Stack Overflow Survey (2024)*.<sup>69</sup> Other reports on DevOps contributors raise a gap in gender diversity. For example, DORA’s *Accelerate State of DevOps Report (2024)* [34] showed that only 12% of DevOps contributors identified themselves as women.

Despite these observations, limited empirical research exists on the composition of DevOps contributors compared to other types of contributors. This gap motivates us to set out to perform an empirical study by using data from 450 active and mature projects that are hosted on GitHub. Below are the RQs that we explore in this chapter, and a preview of the corresponding results.

**(RQ1) Does the perceptible ethnic and gender diversity of DevOps contributors differ from that of non-DevOps contributors?** Since DevOps plays a major role in the software development process [42], a diverse DevOps team would ensure the DevOps process is more robust, inclusive, and successful. Therefore, this RQ aims to provide the community with more awareness of the presence or lack of ethnic and gender diversity of DevOps contributors to open-source projects.

**Results.** With respect to ethnic diversity, contributors with perceptibly White names are the majority among both DevOps contributors (median = 87.70%) and non-DevOps contributors (median = 85.50%). With respect to gender diversity, contributors with names perceived as men are the majority among both DevOps contributors (median = 93.75%) and non-DevOps contributors (median = 92.82%). We statistically measure the ethnic and gender diversity of both DevOps and non-DevOps contributors using diversity metrics (e.g., Blau index [149]), and we find that the diversity of DevOps contributors is significantly less than that of non-DevOps contributors (Wilcoxon signed rank test,  $p < \alpha = 0.0023$ , one-tailed, paired). Note that the Bonferroni-corrected significance level for this chapter is 0.0023 [17].

**(RQ2) How does the distribution of perceptible ethnic and gender diversity change as projects age?** While a concerning lack of ethnic and gender diversity in open-source communities has been reported for decades now [51, 173, 32], it is not yet clear where the current trend is headed. By analyzing the diversity metrics over time, we can better understand whether the trend of diversity is improving or further degrading. Therefore, we examine the evolution of ethnic and gender diversity of DevOps and non-DevOps contributors in the projects.

---

<sup>69</sup><https://survey.stackoverflow.co/2024>

**Results.** As projects evolve, contributors who are perceptibly non-White remain greatly underrepresented. The overall percentage of DevOps contributors who are perceptibly White is decreasing over time; however, the percentage of DevOps contributors who are perceptibly non-White is still low, i.e., it varies between 0%–16.02%. For non-DevOps contributors, the percentage varies between 0%–18.77%. With respect to gender diversity, both DevOps and non-DevOps contributors who are perceptibly men dominate over contributors who are perceptibly women. The percentage of contributors who are perceptibly women varies between 0%–12.5% for DevOps and between 0%–9.48% for non-DevOps.

**(RQ3) How does the intersection of perceptible gender and ethnic diversity differ between DevOps and non-DevOps contributors?** Analyzing ethnicity and gender separately, as in the above RQs, may overlook diversity concerns of contributors who belong to multiple minority groups (e.g., those who are perceptibly non-White women). Indeed, prior studies [65, 137, 165, 5, 116, 132] indicated that individuals who identify themselves as in the intersection of two or more minority groups encounter specific obstacles in [Science, Technology, Engineering, and Mathematics \(STEM\)](#) fields. For example, Black women tend to have less exposure to Software Engineering [137] and face systematic impediments to career mobility [65]. Thus, in this RQ, we perform an intersectional analysis to complement existing studies and provide a more realistic understanding of the current state of perceptible ethnic and gender diversity among DevOps contributors [66].

**Results.** Our findings show that even within the already underrepresented group of perceptibly women contributors, those who are perceived as non-White are greatly underrepresented, suggesting that contributors at the intersection of multiple minority identities (e.g., non-White women) may face compounded barriers to contribute to DevOps roles.

We believe that our study provides empirical evidence that contributes towards a better understanding of the perceptible diversity among contributors in open source. While solutions and strategies have been proposed to increase diversity in open-source projects [51, 173, 32], our results underscore the importance of encouraging open-source communities to foster a more diverse and inclusive environment, considering not only the overall project team but also the different subteams within the project, e.g., DevOps contributors. Our results further highlight the need for targeted inclusion strategies that go beyond isolated dimensions of diversity and address the barriers faced by DevOps contributors with multiple minority identities.

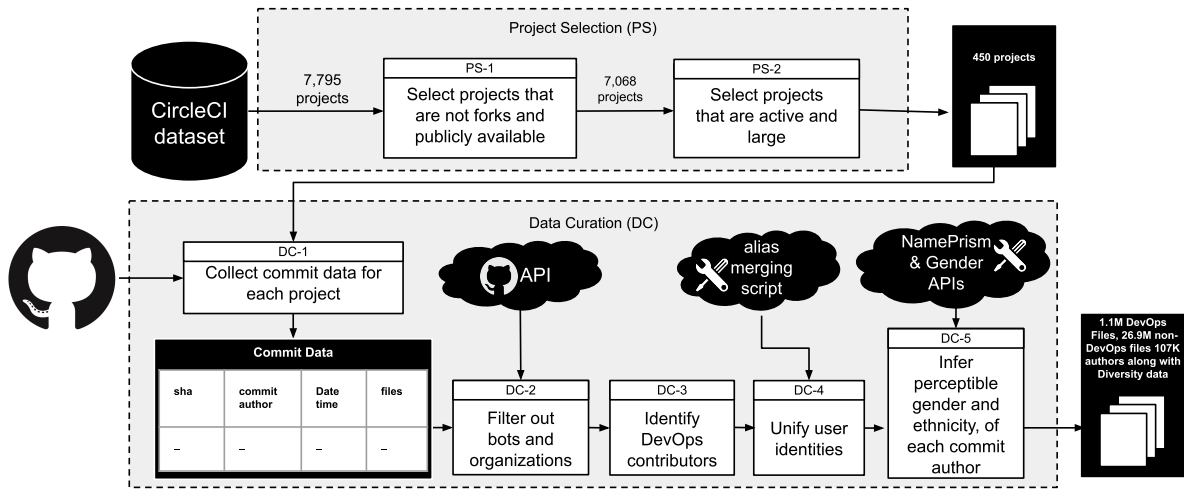


Figure 6.1: An overview of our study design showing project selection (PS) and data curation (DC) steps.

## 6.2 Study Design

In this section, we describe our process for collecting and curating the dataset we use to address our research questions. Fig. 6.1 provides an overview of our study design, which is composed of project selection (PS) and data curation (DC) steps. Below, we explain each step in detail.

### 6.2.1 (PS) Project Selection

Our study aims to analyze the diversity of contributors to DevOps and non-DevOps artifacts in open-source projects. To this end, we need to collect a dataset of open-source projects that adopt tools and technologies for DevOps activities. Fig. 6.1 (PS) provides an overview of our project selection process. We begin with the public dataset of Gallaba et al. [56]. This dataset contains data from 23,330,690 CI builds that span 7,795 GitHub projects that have been using CircleCI,<sup>7</sup> a leading cloud-based CI provider that has served over one million unique contributors during its nine years of operation.<sup>70</sup> This dataset ensures that the projects have potentially been using DevOps tools and technologies to help

<sup>70</sup><https://circleci.com/milestones/>

them automate software development processes, such as building, testing, and deploying the software.

Since GitHub hosts repositories that are not representative of software projects that we aim to investigate (e.g., toy or immature projects) [118], we follow the methodology recommended by previous work [91] to further curate our dataset by applying the following inclusion criteria:

**(PS-1) Select non-forked projects.** GitHub projects can be forks, where a fork is defined as a copy of a project that a GitHub user manages. Forks let GitHub users make changes to a project without affecting the original project. We remove such forks because they largely contain duplicated project history, which would bias our analysis. To do so, we use the GitHub API<sup>71</sup> to determine whether a project is a fork or not. If the project is a fork, the GitHub API returns the `fork` status `True`, and we filter out all such projects. This step reduces our dataset to 7,068 projects.

**(PS-2) Select active and large projects.** Active and large projects are likely to showcase a long-running and collaborative software development process to examine diversity. To detect active and large projects, we consider different thresholds of (1) the number of builds, (2) the number of commits, and (3) the number of contributors.

*Number of builds.* Fig. 6.2 plots candidate threshold values of the number of builds against the number of surviving projects. We select a threshold of 500 builds because it is closer to a “knee” in the curve. Selecting this threshold further reduces the number of projects in the dataset to 2,124. Note that we do not use the Kneedle algorithm [143] (a knee detecting algorithm) to detect knees because such algorithms use a conservative approach to detect knees by identifying a clear inflection point, resulting in an overly conservative number of surviving projects.

*Number of commits.* Fig. 6.3 plots candidate threshold values of the number of commits against the number of surviving projects. We select a threshold of 1,500 commits because it is also closer to a “knee” in the curve. Doing so reduces the number of projects in the dataset to 850 projects.

*Number of contributors.* Fig. 6.4 plots candidate threshold values for the number of contributors against the number of surviving projects. We select a threshold of 50 contributors because we wanted to study projects with a substantial number of contributors. Doing so reduces the number of projects in our dataset to 450.

---

<sup>71</sup><https://docs.github.com/en/rest/repos/repos#get-a-repository>

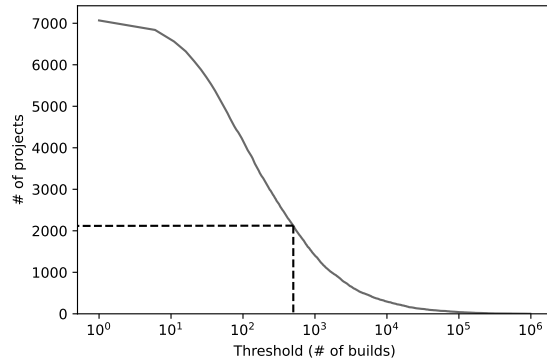


Figure 6.2: Threshold plot for the number of builds.

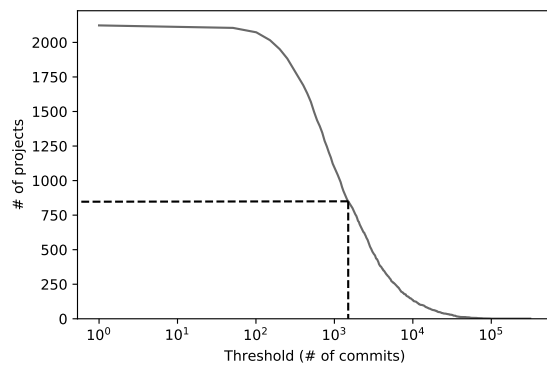


Figure 6.3: Threshold plot for the number of commits.

This dataset of 450 projects comprises large projects from popular organizations, e.g., Facebook,<sup>72</sup> Google,<sup>73</sup> and Angular.<sup>74</sup> Overall, the projects in our dataset run a considerable number of builds (a median of 8,711 builds per project) and have a rich development history (a median of 4,935 commits per project and 145 contributors per project).

---

<sup>72</sup><https://github.com/facebook>

<sup>73</sup><https://github.com/google/mtail>

<sup>74</sup><https://github.com/angular/angular>

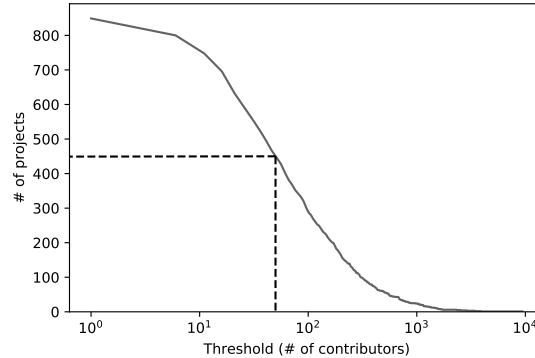


Figure 6.4: Threshold plot for the number of contributors.

## 6.2.2 (DC) Data Curation

In our study, we integrate data from various sources to acquire all the data we need for our analysis. Fig. 6.1 (DC) provides an overview of our data curation process. In the following, we describe each data curation step in detail.

**(DC-1) Collect commit information.** To analyze the diversity of contributors, we need to collect historical data on project development activity. Therefore, we first clone local copies of the 450 repositories in our dataset. Then, for each repository, we mine the commit records that appear on its `master/main` branch (doing so ensures that the most impactful changes to the source code are considered and mitigates the inconsistencies that may arise due to the deletion of temporary branches). For each commit, we extract (meta)data, such as the unique commit ID (i.e., SHA), timestamp, name of the contributor who authored the commit, and files modified (or created) in the commit.

**(DC-2) Filter out commits from non-human contributors.** Our goal is to examine ethnic and gender diversity among contributors in open-source projects, and hence, any contribution made by non-humans (e.g., bots) should not be considered in our data analysis. Therefore, we filter out commits made by non-human contributors. To do so, we use the GitHub API<sup>75</sup> to determine the commit author type, i.e., if the commit is made by a human contributor, the GitHub API returns the type `USER`. We filter out all commits made by authors of a different type. For example, contributions associated with the type `BOT` indicate that the commit is made by a bot (e.g., Dependabot).

<sup>75</sup><https://docs.github.com/en/rest/users/users#about-the-users-api>

**(DC-3) Identify DevOps contributors.** Since we aim to analyze the diversity of contributors to DevOps artifacts, we need to identify potential DevOps artifacts in the studied projects. To do so, we first search for DevOps tools that have the potential to be used in the studied projects. Then, we identify the adoption of such tools in the projects.

**Select DevOps tools.** We opt to select the tools that are used for Deployment, Containers, Builds, and Configuration, in addition to CI, as these practices are frequently integrated into CI pipelines and represent key components of the DevOps toolchain [140]. The table of DevOps tools [1] offered by Digital.ai<sup>76</sup> contains a list of popular DevOps tools used for these practices. For instance, for CI, the listed tools are CircleCI,<sup>7</sup> TravisCI,<sup>9</sup> Jenkins,<sup>6</sup> AWS CodeBuild,<sup>77</sup> and Codeship.<sup>78</sup>

Note that there are other websites that list DevOps tools (e.g., Atlassian,<sup>79</sup> Guru,<sup>80</sup> and Geekflare<sup>81</sup>). We choose the Digital.ai periodic table for two reasons. First, this periodic table offers a comprehensive and organized representation of the DevOps tools [1]. More specifically, it defines 17 categories of tools across five different licensing models from open source through enterprise, covering a wide range of functionalities that cater to diverse IT processes in the DevOps domain. Also, it reflects the votes of over 18,000 DevOps practitioners for over 400 tools. Second, Digital.ai periodic table is widely acknowledged and cited as a reference in academic papers and industry articles in the DevOps field.<sup>82</sup>

**Identify DevOps files.** We begin by inspecting the documentation of each tool to identify the filenames that are used to configure those tools. For example, to configure CircleCI, a `.circleci/config.yml` file is required; any project that contains a commit on the `.circleci/config.yml` file is using (or used) CircleCI service.

A subset of DevOps technologies do not have a filename convention. For example, Ansible<sup>83</sup> and Kubernetes<sup>84</sup> are configured using `.yaml` files, but the team may choose to name the file however they see fit. Since `.yaml` is a popular extension used for all sorts of files, we cannot rely solely on the extension to identify DevOps files. Therefore, two inspectors inspect a sample of 400 `.yaml` files from our dataset that are not explicitly classified as

---

<sup>76</sup><https://digital.ai>

<sup>77</sup><https://aws.amazon.com/codebuild>

<sup>78</sup><https://www.cloudbees.com/products/codeship>

<sup>79</sup><https://www.atlassian.com/devops/devops-tools>

<sup>80</sup><https://www.guru99.com/devops-tools.html>

<sup>81</sup><https://geekflare.com/devops-tools>

<sup>82</sup><https://www.uktech.news/technology-news/digital-ai-releases-new-version-of-industry-standard-periodic-table-of-devops-tools>

<sup>83</sup><https://www.ansible.com>

<sup>84</sup><https://kubernetes.io>

DevOps files based on the filename. The inspectors prepare a list of keywords found in the filename or file path that make a file a *DevOps file* or a *non-DevOps file*. For example, `devops`, `deployment`, `kubernetes`, and `chef` are examples of keywords in a file path, which indicate that `.yaml` file is a *DevOps file*; in contrast, if the file path contains `changelog`, `docs/`, `package`, or `pullapprove`, then that `.yaml` file is a *non-DevOps file*. During this coding, the inspectors also prepare a list of keywords by inspecting the content of a `.yaml` file. For example, any `.yaml` file related to Kubernetes must contain `kind` and `apiVersion` key settings.

After listing the potential keywords, we identify the `.yaml` files that are likely to be DevOps files by matching keywords to the file paths. For the `.yaml` files that are not easily classifiable, we parse the content in search of key settings that we identified as DevOps-related above. After applying these automatic classification steps, 3.4% of the `.yaml` files remain ambiguous. To ensure the integrity of our categories, we remove the ambiguous files from our analysis.

After that, we evaluate our classifier. To do so, a coder manually classifies a sample of 400 `.yaml` files from our dataset. Then, we calculate the Cohen's Kappa coefficient [28] of agreement between the coder and our classifier. This coefficient is commonly used to evaluate inter-rater agreement for categorical items between two raters. The value of Cohen's Kappa coefficient ranges from -1 to +1, with values greater than zero indicating an agreement better than chance. We obtain a coefficient of 0.82, indicating near-perfect strength of agreement between the coder and the classifier. The false positive rate is 0.005, meaning that 0.5% of negative instances are falsely identified as positive, while the false negative rate is 0.227, indicating that 22.7% of positive instances are incorrectly labelled as negative.

**Clean files dataset.** Through a preliminary inspection of the files in our dataset, we observe that a non-negligible proportion is not relevant to the purpose of our analysis, e.g., `node_modules/node-inspector/node_modules/v8-debug/node_modules/node-pre-gyp/node_modules/mkdirp/package.json` and `vendor/k8s.io/kubernetes/cmd/mungedocs/links.go` are automatically generated files that are not maintained by hand. Thus, two inspectors examine a sample of 200 files from our dataset and prepare a list of keywords in a file path that makes a file an auto-generated dependency file or not. For example, the files within the folders named `vendor` and `node_modules` contain third-party dependencies. After listing the potential keywords, we identify the files that are likely to be dependency files by matching keywords to the file paths. The percentage of such files amounts to 15.7% of all the files in our dataset, and we remove them from our analysis to ensure the validity of the results.

To evaluate the aforementioned classifier, a coder manually classifies a sample of 400 files from our dataset. We then calculate the Cohen’s Kappa coefficient of agreement between the coder and our classifier. We obtain a Kappa coefficient of 0.81, showing near-perfect agreement between the coder and the classifier. The false positive rate is 0.056, meaning that 5.6% of negative instances are falsely identified as positive, while the false negative rate is 0.047, indicating that 4.7% of positive instances are incorrectly labelled as negative.

Then, for each project, we include in our dataset the commits made after the first DevOps-related commit in the project and until the end of the year 2021. Analyzing the commits after the first DevOps-related commit ensures that the timeframes for non-DevOps and DevOps artifacts align.

***Finalize DevOps and non-DevOps contributors.*** Once we identify the DevOps files, for each contributor in our dataset, we check for the type of files changed in the commits made by the contributor. We find that the majority of the contributors in a project (median = 86.70%) only contribute to non-DevOps files, which we refer to as non-DevOps contributors. However, a non-negligible percentage (median = 13.30%) of contributors made at least one commit to DevOps files. We refer to such contributors as DevOps contributors.

**(DC-4) Unify identities.** On GitHub, commits may not be attributed properly due to variations in a committer’s name and email address.<sup>85</sup> For example, consider the email address `sandyw@gmail.com` that could be associated with two names based on the local configurations of the contributor: `Sandy W` and `Sandy White`. Besides, the same contributor may use different email addresses. For example, `Sandy White` may use their personal email `sandyw@gmail.com` as well as the noreply email address (`sandy@users.noreply.github.com`) as their commit-email address.<sup>86</sup> In order to unify such cases with different identities of the same GitHub contributor, we use the `GitHub-alias-merging` script by Vasilescu et al. [169]. This script uses heuristics to link different aliases and email addresses belonging to the same GitHub contributor. By using this script, we identify 110,336 unique contributors from 138,012 `<name,email>` pairs in our original dataset. For the rest of the study, we use these unique contributor identities.

**(DC-5) Infer perceptible demographics of contributors.** Since we aim to investigate gender and ethnic diversity, we need to classify contributors accordingly. Since gender and

---

<sup>85</sup><https://git-scm.com/book/en/v2/Git-Basics-Git-Aliases>

<sup>86</sup><https://docs.github.com/en/account-and-profile/setting-up-and-managing-your-personal-account-on-github/managing-email-preferences/setting-your-commit-email-address>

ethnic identity are difficult to ascertain at scale, similar to prior work [120, 168, 119, 135], we rely on perceived identity characteristics that can be inferred based on publicly visible profile data.

***Infer perceptible genders.*** It is not our aim to establish new means of inferring gender. There already exist open-source tools that are capable of inferring gender based on the name of the contributor, such as `Gender-guesser`,<sup>87</sup> `GenderComputer`,<sup>88</sup> and `Wiki-Gendersort`.<sup>89</sup> Unfortunately, these tools are not without limitations. For example, Santamaría et al. [142] evaluated the `Gender-guesser` tool and found that the tool produced a rate of 20.12% unrecognized names [142]. In such cases, the tool predicts the gender as `None`; for example, the gender predicted from the names, such as `rd`, `ga`, and `xulin` was `None`. To tackle the problem of the high rate of unrecognized names, inspired by prior work [145], we combine the outcomes from all three gender-inferring tools and evaluate the rate of unrecognized names. Among the different combinations we study, the most effective combination to infer gender for the test dataset (provided by Santamaría et al. [142]) is first to infer the gender using `Wiki-Gendersort`, and if the gender is unrecognized, then infer the gender using `Gender-guesser` and `GenderComputer`. Doing so reduces the rate of unrecognized names (20.12%) to 5.15%. Thus, we used the aforementioned combination of tools to infer the perceptible genders of contributors in our dataset. We discard the names that none of the tools are able to infer the perceptible gender from our analysis.

***Infer perceptible ethnicities.*** Inspired by prior work on diversity [120, 119, 67, 45], we use the NAME-PRISM [182] tool to infer the perceptible ethnicity of contributors. NAME-PRISM [182] is a name-based perceptible ethnicity classification tool, which uses name-embeddings to predict the ethnicity of a person by using their name. This classifier was trained using United States (US) Census Bureau data,<sup>90</sup> and it predicts the probabilities of a given name belonging to a person of White, Black, Asian/Pacific Islander (API), Hispanic, American Indian/Alaskan Native (AIAN), and Mixed Race (2PRACE). A recent study by Preoțiuc-Pietro et al. [133] evaluated the performance of the NAME-PRISM’s [182] ethnicity classifier. They measured the AUROC of NAME-PRISM with respect to the top four ethnic groups: White, API, Hispanic, and Black. Their evaluation showed that the AUROC of classifying people from White, API, Hispanic, and Black were 0.719, 0.765, 0.740, and 0.681, respectively. To strengthen the validity of our study, we discard the names where the ethnicity is unknown from our ethnic diversity analysis; we find only 0.03% (five) and 0.06% (59) names in which the ethnicity is unknown among DevOps and

---

<sup>87</sup><https://pypi.org/project/gender-guesser>

<sup>88</sup><https://github.com/tue-mdse/genderComputer>

<sup>89</sup><https://github.com/nicolasberube/Wiki-Gendersort>

<sup>90</sup><https://www.census.gov/quickfacts/fact/note/US/RHI625221>

Table 6.1: Descriptive statistics of our curated dataset of 450 projects.

Metric	Min	Median	Max	Total
The number of contributors	45	164	4,033	110,336
The number of commits	1,034	4,653	151,416	4,216,386
The number of DevOps file changes	6	479	161,825	1,197,829
The number of non-DevOps file changes	2,454	26,930	813,974	29,750,595
The number of perceptibly men	24	120	3,164	104,883
The number of perceptibly women	0	10	297	9,166
The number of perceptibly White contributors	14	116	2,721	96,554
The number of perceptibly Hispanic contributors	0	4	138	3,671
The number of perceptibly API contributors	0	15	542	15,197
The number of perceptibly Black contributors	0	0	22	495
The number of perceptibly AIAN contributors	0	0	1	2

Table 6.2: A sample of our curated dataset.

Project Owner	Project Name	sha	Datetime	File	Is Dev Ops File?	Author	Perceived Gender	Perceived Ethnicity
angular	angular	ef6...	2021-5-19 12:12:32.345237	.circleci /config.yml	True	Kavya	woman	API
angular	angular	c68...	2021-5-19 12:16:01.459565	docs/COMM ITTER.md	False	Pete Clay	man	White
angular	angular	c68...	2021-5-19 12:16:01.459565	docs/PUB LIC_API.md	False	Pete Clay	man	White

non-DevOps contributors, respectively. Furthermore, following prior studies that rely on the US Census Bureau’s classification to label developers across different countries and contexts [120], we only use contributors’ names where the perceptible ethnicity is inferred by NAME-PRISM [182] with a confidence level greater than 0.8.

Table 6.1 shows the descriptive statistics of our final dataset. It contains 1,197,829 DevOps file changes and 29,750,595 non-DevOps file changes made by 110,336 contributors to 450 GitHub projects, and an anonymized version of this dataset is available online along with our replication package.<sup>91</sup> Table 6.2 shows a sample of data from this dataset. The data items in this preview table are altered to protect the contributors’ identities.

<sup>91</sup><https://doi.org/10.5281/zenodo.8277702>

## 6.3 Study Results

In this section, we describe our approach and then present our results for each RQ.

### 6.3.1 (RQ1) Does the perceptible ethnic and gender diversity of DevOps contributors differ from that of non-DevOps contributors?

In this RQ, we quantitatively analyze the perceptible ethnic and gender diversity of DevOps and non-DevOps contributors in the studied projects.

#### Approach

We first compute the percentages of DevOps and non-DevOps contributors perceived as different ethnicities and genders. Then, we statistically test the significance of the difference between the percentages of DevOps contributors and non-DevOps contributors. In particular, we use the Wilcoxon signed rank test with a 95% confidence level. We use this non-parametric test because our data are not normally distributed.

To complement the previous analysis, we compute other common metrics, including the Richness, Evenness [37, 20], Blau index (a.k.a. Diversity index/Simpson index) [149], and Prevalence rankings and Diffusion score,<sup>92</sup> which are used to measure the diversity in a community. These metrics have been used in prior studies for similar purposes [22, 126, 189, 168]. The first metric, richness ( $R$ ), measures the number of groups in a community [37]. We use the richness metric to measure the number of ethnicities of contributors to DevOps and non-DevOps files in a project. The higher the richness, the more diverse the community is. For example, if a project contains only contributors with perceptibly White names, the ethnic richness of that project is one. If a project contains contributors of two perceptible ethnicities (e.g., White and Asian), the richness value is two. Similarly, for gender diversity, if a project includes contributors who are perceived as men only, the richness is one. If a project includes contributors who are perceived as men as well as those who are perceived as women, the richness is two. The second metric, evenness ( $E$ ), measures the relative species abundances in a community. We use the Brillouin metric [20] to measure the evenness. Equation (6.1) shows the formula for computing the evenness in a community,

---

<sup>92</sup><https://www.census.gov/newsroom/blogs/random-samplings/2021/08/measuring-racial-ethnic-diversity-2020-census.html>

where  $n_i$  is the number of individuals in a group  $i$ , and  $N$  is the total number of individuals in the community.

$$E = \frac{\ln(N!) - \sum_{i=1}^S \ln(n_i!)}{N} \quad (6.1)$$

In the context of our study, suppose a project contains 10 contributors ( $N = 10$ ) spanning four ethnicities ( $R = 4$ ): three perceptibly Asian contributors ( $n_1 = 3$ ), two perceptibly Black contributors ( $n_2 = 2$ ), one perceptibly Hispanic contributors ( $n_3 = 1$ ), and four perceptibly White contributors ( $n_4 = 4$ ). The evenness is computed as follows:  $\frac{\ln(10!) - (\ln(3!) + \ln(2!) + \ln(1!) + \ln(4!))}{10} = 0.94$ . This value shows a high level of evenness, indicating a high level of ethnic diversity of contributors. Similarly, the closer this number is to zero, the lower the evenness, thus the lower the diversity. Following the same equation, we compute the evenness for gender diversity as well.

The next metric, the Blau index, a.k.a diversity index (D) [149], computes the probability that two individuals randomly selected from a community would belong to two different groups within the community. Equation (6.2) shows the formula to compute the Blau index of a community. Blau index ranges from zero to one. The closer the Blau index is to one, the more diverse the community is.

$$D = 1 - \sum_{i=1}^S \left(\frac{n_i}{N}\right)^2 \quad (6.2)$$

In the previous example with 10 contributors, the Blau index is computed as below:  $1 - \left(\left(\frac{3}{10}\right)^2 + \left(\frac{2}{10}\right)^2 + \left(\frac{1}{10}\right)^2 + \left(\frac{4}{10}\right)^2\right) = 0.7$ ; accordingly, there is high probability of two contributors randomly selected from the project belong two different ethnicities, thus indicating a high level of ethnic diversity in the project. Similarly, we compute the Blau index for gender diversity as well.

Lastly, we compute the prevalence rankings and diffusion scores.<sup>92</sup> The prevalence ranking in a community is the ranking of groups in descending order of the number of individuals belonging to each group. From the prevalence rankings of a community, first-, second-, and third-prevalent groups could be obtained. The percentage of the individuals that are not in those first-, second-, or third-prevalent groups combined is called the diffusion score. The first-, second-, and third-prevalent ranks in the previous example are owned by perceptibly White, Asian, and Black contributors, with proportions of 40%, 30%, and 20%, respectively. Thus, the diffusion score is  $100\% - (40\% + 30\% + 20\%) = 10\%$ , which is equal to the percentage of least-prevalent ethnicity (Hispanic) in this example since there

are only four ethnicities. For gender diversity in the context of our study, the diffusion score will be equal to the contributor percentage from the minority gender. This is because we consider a binary classification of gender, and accordingly, there are only first- and least-prevalent groups with respect to the perceptible gender. Thus, we consider the percentage of contributors from the least-prevalent gender in a particular project/setting as the diffusion score of that case.

To test the statistical difference between diversity metrics for DevOps and non-DevOps, we first compute the metrics for each project. Then, to compare the metrics of DevOps and non-DevOps contributors, we perform the Wilcoxon signed rank test with a 95% confidence level ( $\alpha = 0.05$ ). To account for multiple comparisons, we apply the Bonferroni correction [17]—a widely used method for adjusting the significance level to reduce the likelihood of false positives. As our study involves 22 statistical comparisons, the adjusted significance level is set to  $\frac{0.05}{22} \approx 0.0023$ .

## Results

Below, we present the results of the above analysis and highlight the key observations.

**Observation 6(1): Contributors to DevOps artifacts tend to have less ethnic diversity than contributors to non-DevOps artifacts.** Fig. 6.5 shows the distribution of the percentage of contributors to DevOps and non-DevOps artifacts in the studied projects, per ethnicity. From the figure, we observe that contributors who are perceived as White are the majority of DevOps contributors (median = 87.70%) and non-DevOps contributors (median = 85.50%).

The second dominating ethnicity in our dataset is Asian/Pacific Islander (API). On median, the percentages of DevOps and non-DevOps contributors with perceptibly API names are 9.10% and 10.53%, respectively. We find a statistically significant difference between the percentages of DevOps and non-DevOps contributors with perceptibly API names but with a negligible effect size (Wilcoxon,  $p \ll \alpha = 0.0023$ , one-tailed, paired; Cliff’s  $|\delta| = 0.132$ ). The third dominating ethnicity is Hispanic. On median, the percentages of DevOps contributors with perceptibly Hispanic names are 0% and 3.25%, respectively. We find a statistically significant difference between DevOps and non-DevOps contributors with a medium effect size (Wilcoxon,  $p \ll \alpha = 0.0023$ , one-tailed, paired; Cliff’s  $|\delta| = 0.415$ ). The fourth dominating group is the contributors having perceptibly Black names. For those contributors, we find a statistically significant difference between the percentages of DevOps (median = 0%) and non-DevOps (median = 0.77%) contributors with a large effect size (Wilcoxon,  $p \ll \alpha = 0.0023$ , one-tailed, paired; Cliff’s  $|\delta| =$

0.783). Finally, we find that the contributors with perceptibly American Indian/Alaskan Native (AIAN) names are the least dominating ethnic group among DevOps contributors and non-DevOps because only two out of 450 projects we use for the analysis has contributors perceived as AIAN. Moreover, none of the DevOps contributors in those two projects are perceived as AIAN.

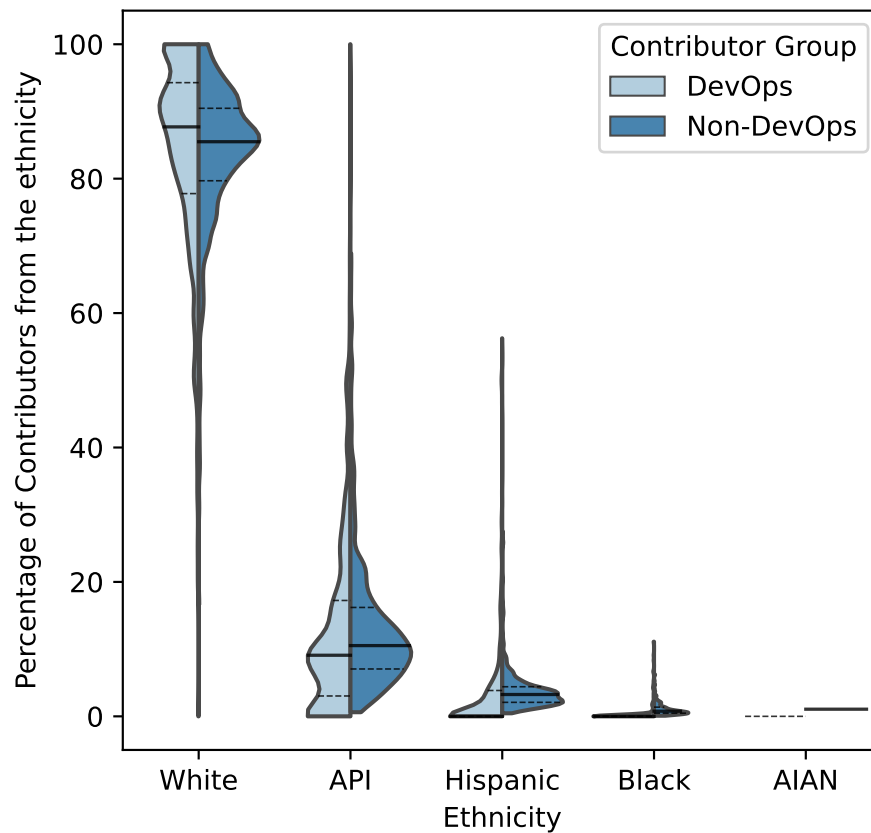


Figure 6.5: Bean plots showing the distribution of percentages of DevOps and non-DevOps contributors from four perceptible ethnicities (i.e., White, API, Hispanic, and Black). The solid lines represent the median percentages, and the dotted lines represent the first and third quartiles.

Table 6.3: Results of statistical analysis of ethnic diversity metrics for DevOps and Non-DevOps contributors.

Diversity Metric	DevOps (Median)	Non-DevOps (Median)	Wilcoxon, One-Tailed, Paired ( $\alpha = 0.0023$ )	Effect Size (Cliff's $ \delta $ )
Richness	2	3	$p = 2.81 \times 10^{-47} \ll \alpha$	0.588, large
Evenness	0.330	0.467	$p = 2.41 \times 10^{-31} \ll \alpha$	0.246, medium
Blau Index	0.219	0.255	$p = 6.95 \times 10^{-7} \ll \alpha$	0.111, negligible
Diffusion Score	9.091	10.157	$p = 0.006 > \alpha$	-

Furthermore, Table 6.3 presents the results of the analysis of ethnic diversity metrics for DevOps and non-DevOps contributors. This table shows that the median values of ethnicity richness of DevOps and non-DevOps contributors are two and three, respectively. Also, we find statistically significant differences in terms of richness of ethnic diversity between DevOps and non-DevOps contributors, with a large effect size. Similarly, the ethnic evenness of DevOps contributors is significantly less than that of non-DevOps contributors, with a medium effect size. For the Blau index, we find statistically significant differences between the ethnic diversity of DevOps and non-DevOps contributors, but the effect size is negligible. Also, note that for the diffusion score, we do not find statistical significance between DevOps and non-DevOps contributors.

**Observation 6(2): The perceptible gender diversity of DevOps contributors tends to be less than that of non-DevOps contributors.** Fig. 6.6 shows the distribution of percentages of DevOps and non-DevOps contributors who are perceived as men and women. Accordingly, the DevOps contributors who are perceived as men are the majority of DevOps contributors (median = 93.75%) as well as non-DevOps contributors (median = 92.82%). We find a statistically significant difference between the percentages of DevOps and non-DevOps contributors who are perceived as men, with a small effect size (Wilcoxon,  $p \ll \alpha = 0.0023$ , one-tailed, paired; Cliff's  $|\delta| = 0.149$ ).

Furthermore, the median percentage of DevOps contributors who are perceived as women (median = 6.25%) is less than that of non-DevOps contributors (median = 7.18%). We further find a statistically significant difference between DevOps and non-DevOps contributors who are perceived as women, with a small effect size. This is in line with the data collected from the *Stack Overflow Survey (2022)*,<sup>17</sup> which shows that the percentage of DevOps developers who identify themselves as women is 2.10%, while the other developers, i.e., non-DevOps, who identify themselves as women, is 5.13%.

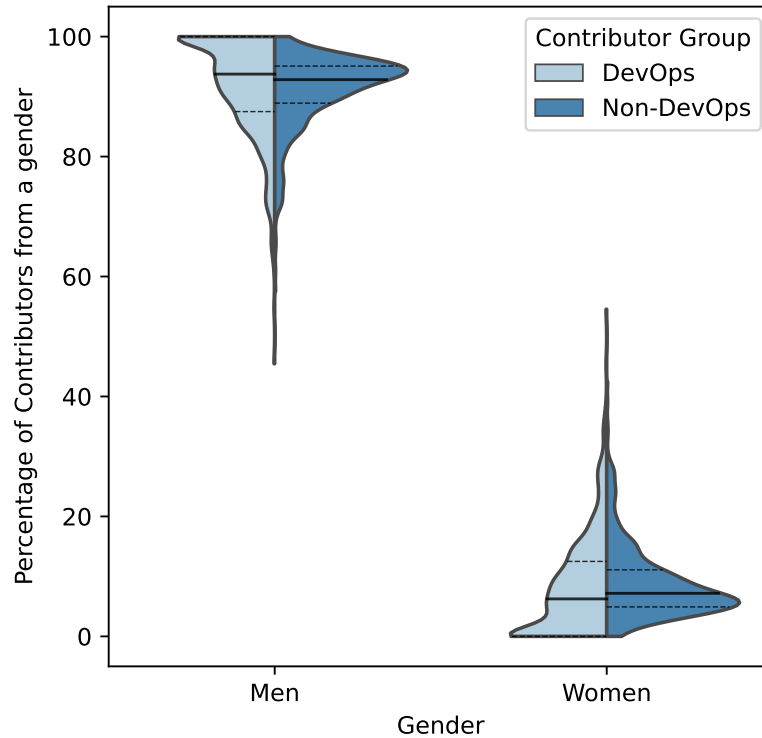


Figure 6.6: Bean plots showing the distribution of percentages of contributors perceived as men and women among DevOps contributors as well as among non-DevOps contributors. The solid lines present the median, and the dotted lines present the first and third quartiles.

Table 6.4 reports the results of our analysis of gender diversity among DevOps and non-DevOps contributors. It shows that the median value of richness in terms of gender is two for both DevOps and non-DevOps contributors; however, we find a statistically significant difference in gender richness between DevOps and non-DevOps contributors, with a small effect size. Similarly, the evenness, Blau index, and diffusion score of DevOps contributors in projects, in terms of gender diversity, are statistically less than those of non-DevOps contributors, with a small effect size. Furthermore, we find that 29.11% of the projects in our dataset did not contain any DevOps contributors perceived as women. In contrast, only 2.00% of the projects in our dataset did not contain non-DevOps contributors who are perceived as women.

Table 6.4: Results of statistical analysis of gender diversity metrics for DevOps and Non-DevOps contributors.

Diversity Metric	DevOps (Median)	Non-DevOps (Median)	Wilcoxon, One-Tailed, Paired ( $\alpha = 0.0023$ )	Effect Size (Cliff's $ \delta $ )
Richness	2	2	$p = 1.44 \times 10^{-25} \ll \alpha$	0.271, small
Evenness	0.181	0.243	$p = 7.79 \times 10^{-20} \ll \alpha$	0.246, small
Blau Index	0.117	0.133	$p = 1.58 \times 10^{-6} \ll \alpha$	0.149, small
Diffusion Score	6.250	7.176	$p = 6.44 \times 10^{-5} \ll \alpha$	0.149, small

Note that we exclude contributors whose perceptible gender is not determined by gender-inferring tools, as mentioned in Section 6.2.2 (DC-5). However, studying the effects of perceiving all unknown genders as either men or women can provide valuable insights into gender representation and potential biases in various contexts. Thus, we follow the approach of Vasilescu et al. [167], who faced a similar issue. In particular, we investigate the impact of assuming all unknown genders as women. Since women contributors are typically underrepresented in GitHub [168], assuming all unknown genders as women would allow us to evaluate whether biases persist even when we artificially increase the representation of women contributors. From this analysis, we observe that our findings still hold: those who are perceived as women are more underrepresented among DevOps contributors than non-DevOps contributors. A detailed preview of our results, assuming all contributors with unknown genders are women, is available in our Online Appendix B.<sup>91</sup>

As for perceptible ethnic diversity, we observe that contributors perceived as non-White (API, Hispanic, Black, and AIAN) are more underrepresented among DevOps contributors compared to non-DevOps contributors. With respect to perceptible gender diversity, we find contributors perceived as women are more underrepresented among DevOps contributors compared to non-DevOps contributors. Overall, there is a statistically significant difference in the diversity metrics between DevOps and non-DevOps contributors.

### 6.3.2 (RQ2) How does the distribution of perceptible ethnic and gender diversity change as projects age?

In this section, we present the approach that we used to examine the evolution of gender and ethnic diversity among DevOps and non-DevOps contributors in the studied projects, along with the corresponding results.

#### Approach

We begin by partitioning the commits in our dataset into segments, where each segment represents a set of commits that are made during a year. Then, for each segment (year), we identify DevOps and non-DevOps contributors who contributed to the commits made during that year and analyze perceptible ethnic and gender diversity. In particular, we examine the yearly percentages of ethnicities and genders, followed by the computation of diversity metrics for each corresponding year. Then, we compute the average growth in diversity metrics over the last ten years. Equation (6.3) shows how we compute the average growth, for example, in evenness, for the last ten years where  $n = 10$ .

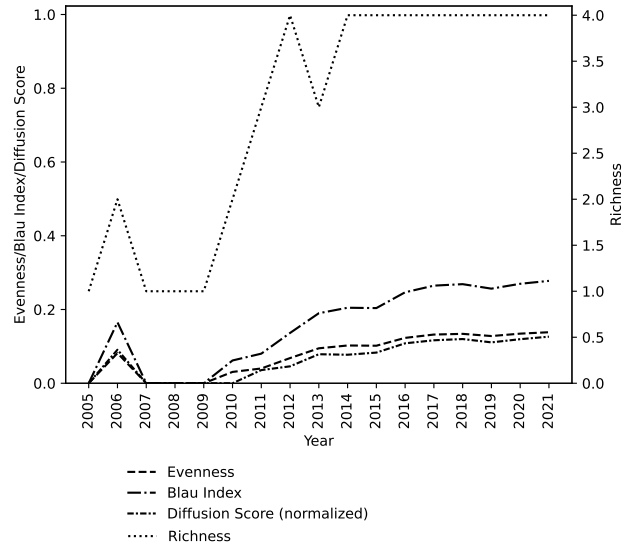
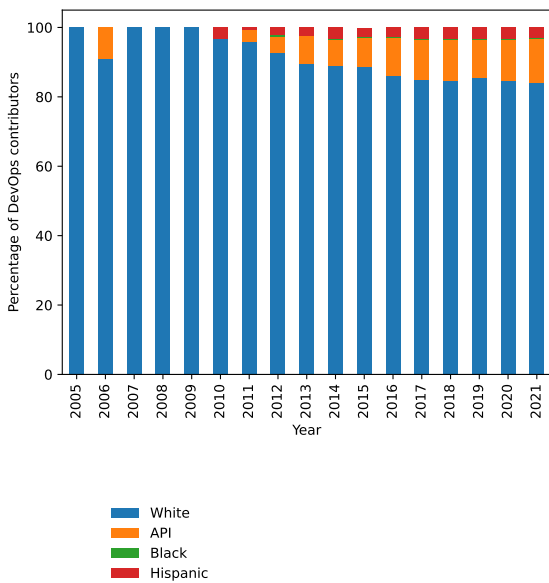
$$\text{Average growth} = \sum_{t=1}^{10} \text{evenness}_t - \text{evenness}_{t-1} \quad (6.3)$$

#### Results

Below, we discuss the observations that we made.

**Observation 6(3): Over time, the overall perceptible ethnic diversity of DevOps and non-DevOps contributors increases. Still, the contributors perceived as non-White (API, Hispanic, and Black) are the minority.** Fig. 6.7 shows (a) the percentage of DevOps contributors from each perceptible ethnicity per year and (b) the values of the diversity metrics per year. From Fig. 6.7a, we observe that the percentage of DevOps contributors who are perceived as White generally varies between 83.98%–100%.

Also, we observe that all DevOps contributors who made commits in the studied projects before 2010 are perceived as White (except for the year 2006). Hence, the evenness metric of these years is zero (Fig. 6.7b). From 2010 onward, DevOps contributors with perceptibly non-White names have started to participate in the studied projects; however, the percentage remains low. For example, the percentage of DevOps contributors with API names ranges from 0%–12.63%, and the percentage of contributors with Hispanic names



(a) Percentages of perceptible ethnicities of DevOps contributors over time.

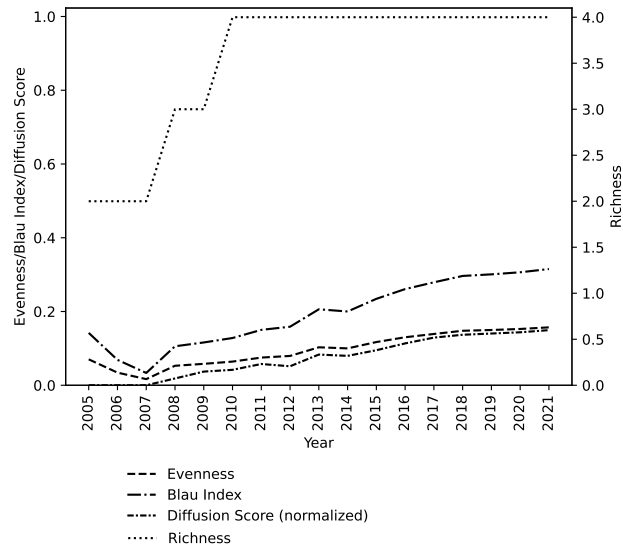
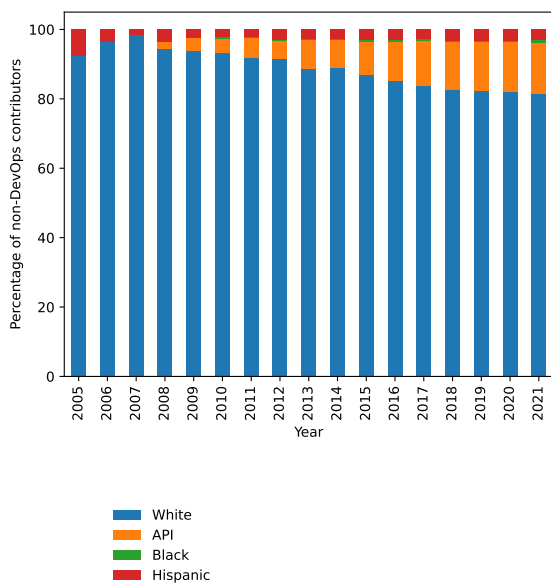
(b) Diversity metrics over time: Both the left and right y-axes present the metrics values. The diffusion score is normalized.<sup>93</sup>

Figure 6.7: Change in the perceptible ethnic diversity of DevOps contributors over time.

ranges from 0%–3.30%. This is further evident in Fig 6.7b. The figure shows that the diversity metrics for the ethnicity of DevOps contributors are increasing over time, yet the evenness, Blau index, and diffusion score mostly remain constant over the last ten years. In particular, the evenness ranges from 0.068–0.138, and its average growth is 0.008 over the last ten years; the Blau index ranges from 0.136–0.278 with an average growth of 0.016; similarly, the normalized diffusion score<sup>93</sup> ranges from 0.045–0.126, and its average growth is 0.009 over the last ten years.

The diversity of non-DevOps contributors belonging to different ethnicities follows a similar trend to that of DevOps contributors. Fig. 6.8 shows that diversity metrics for the ethnicity of non-DevOps contributors are gradually increasing over time. We observe that the ethnic evenness, Blau index, and diffusion score for non-DevOps contributors range between 0.079–0.157, 0.158–0.315, and 0.051–0.149, respectively, over the last ten years; their average growth over time are 0.009, 0.017, and 0.010, respectively.

<sup>93</sup>The diffusion score is usually a percentage, but for visualization purposes, we normalize the diffusion score by dividing the score by 100 to show the corresponding proportion.



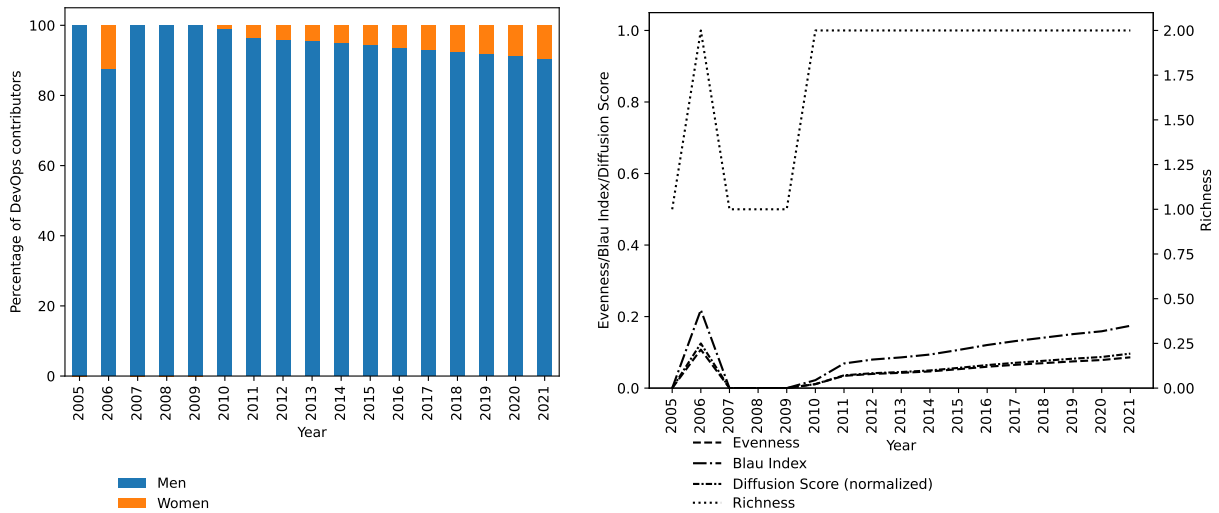
(a) Percentages of perceptible ethnicities of non-DevOps contributors over time.

(b) Diversity metrics over time: Both the left and right y-axes present the metrics values. The diffusion score is normalized.<sup>93</sup>

Figure 6.8: Change in the perceptible ethnic diversity of non-DevOps contributors over time.

Note that the above observation discussing the increase of ethnic diversity over time is in line with the diversity analysis of US-based DevOps job-seekers by Zippia<sup>16</sup> which is a web application that is used to search US-based jobs. Zippia’s analysis revealed that the percentage of non-White developers increased from 37.16% in 2010 to 42.77% in 2021.

**Observation 6(4): Over time, the perceptible gender diversity of DevOps and non-DevOps contributors is increasing. Still, the contributors who are perceived as women remain underrepresented.** Figures 6.9 and 6.10 show (a) the percentage of perceived genders of DevOps and non-DevOps contributors and (b) the change in diversity metrics over time. We observe that the percentage of DevOps and non-DevOps contributors who are perceived as men varies between 87.50%–100% and between 90.52%–100%, respectively. Moreover, we observe that the overall richness, evenness, Blau index, and diffusion score for DevOps and non-DevOps contributors have been increasing over time, yet the improvement is gradual. For example, the evenness for DevOps contributors over the past ten years ranges from 0.040–0.086, while it is 0.047–0.085 for non-DevOps contributors. In addition, we find that the average growth in evenness is 0.005 for DevOps



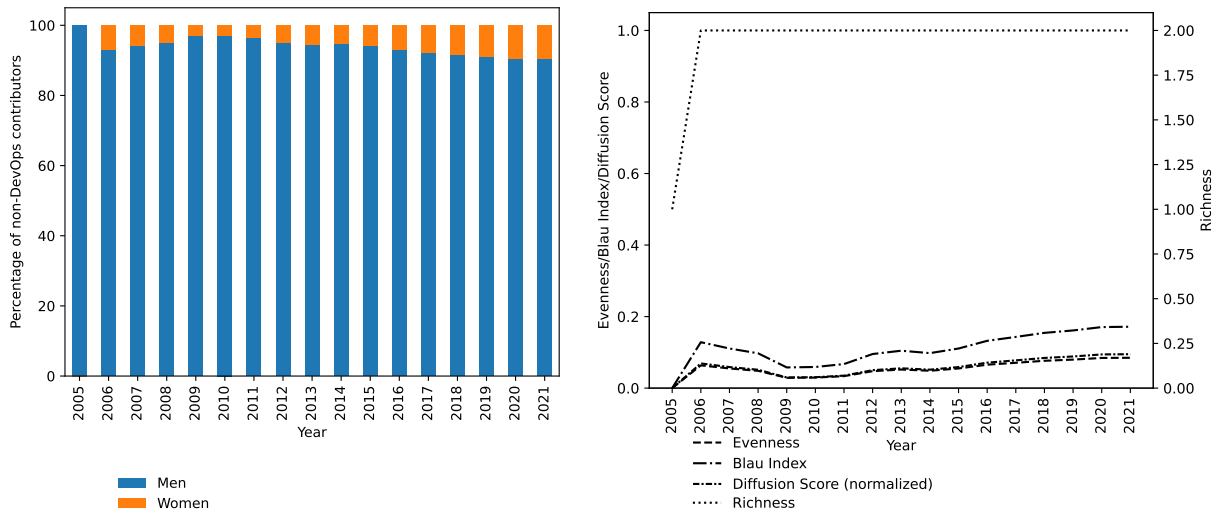
(a) Percentages of perceptible gender of DevOps contributors over time. (b) Values of diversity metrics over time: Both the left y-axis and the right y-axis present the metrics values. Note that the diffusion score is normalized.<sup>93</sup>

Figure 6.9: Change in the perceptible gender diversity of DevOps contributors over time.

contributors and 0.004 for non-DevOps contributors. We observe similar trends for the Blau index and diffusion score as well. This observation complements Prana et al.’s [132] study, which found that gender diversity in open-source projects (GHTorrent<sup>94</sup> dataset) has increased over time (2014–2018); however, there is still much room for improvement.

Finally, similar to the sensitivity analysis of gender diversity in RQ1, for RQ2, we investigate the impact of assuming all contributors with unknown genders as women [167]; however, we do not find a substantial change in our Observation 6(4). A detailed preview of our results, assuming all unknown genders are women, is available in our Online Appendix B.<sup>91</sup>

<sup>94</sup><http://ww38.ghtorrent.org>



(a) Percentages of perceptible gender of non-DevOps contributors over time.

(b) Values of diversity metrics over time: Both the left y-axis and the right y-axis present the metrics values. Note that the diffusion score is normalized.<sup>93</sup>

Figure 6.10: Change in the perceptible gender diversity of non-DevOps contributors over time.

Although the perceptible ethnic diversity of contributors is increasing over time, DevOps and non-DevOps contributors with perceptibly non-White names are greatly underrepresented. Similarly, the perceptible gender diversity of contributors is gradually increasing over time. Still, contributors who are perceived as women remain substantially underrepresented.

### 6.3.3 (RQ3) How does the intersection of perceptible gender and ethnic diversity differ between DevOps and non-DevOps contributors?

Our study thus far has been performed to independently analyze the perceptible gender and ethnic diversity of DevOps and non-DevOps contributors. However, it is unclear how diverse DevOps and non-DevOps contributors are when considering the intersection of minority groups of perceptible ethnicity and gender (e.g., perceptibly women contributors

with perceptibly non-White names). In fact, prior studies [65, 137, 165, 5, 116, 132] have shown that individuals who belong to the intersection of two minority groups face specific challenges in STEM fields. Indeed, contributors who belong to two or more underrepresented categories tend to receive fewer opportunities [65].

In this RQ, we shed light on the intersectionality of gender and ethnic diversity of DevOps and non-DevOps contributors. Below, we describe our approach to examine the participation of contributors with perceptibly non-White and women names, along with the corresponding results.

## Approach

We first compute the percentage of DevOps and non-DevOps contributors perceived as women of different perceptible ethnicities. Then, we test the significance of the difference between such DevOps and non-DevOps contributors in the studied projects using the Wilcoxon signed rank test with a 95% confidence level (with Bonferroni correction;  $\alpha = 0.0023$ ). To examine the trend of intersectionality over time, we then compute the diversity metrics for DevOps and non-DevOps contributors per year.

## Results

Below, we present the observations derived from the above analysis in response to this RQ.

**Observation 6(5): Among DevOps and non-DevOps contributors who are perceptibly women, those who are perceived as White are the majority.** Table 6.5 reports the percentage of perceptible ethnicities of perceptibly women DevOps and non-DevOps contributors. Since none of the DevOps contributors in our dataset are perceived as AIAN, we do not consider the AIAN in this intersectional analysis. This table shows that contributors with perceptibly White names are the majority among perceptibly women DevOps contributors (median = 100%) and non-DevOps contributors (median = 70%) in a project. The second ethnic majority among perceptibly women contributors is API; the median percentages of DevOps and non-DevOps contributors are 0% and 25%, respectively. Perceptibly women contributors with perceptibly Hispanic and Black names are the least included among DevOps and non-DevOps contributors (median = 0%).

Table 6.5 also shows statistically significant differences in the perceived ethnic diversity between DevOps and non-DevOps contributors with perceptibly women names. In particular, the contributors who are perceived as White women are more represented in DevOps

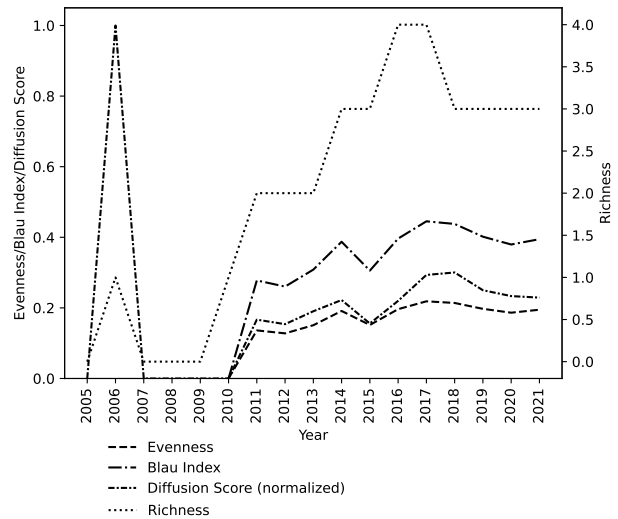
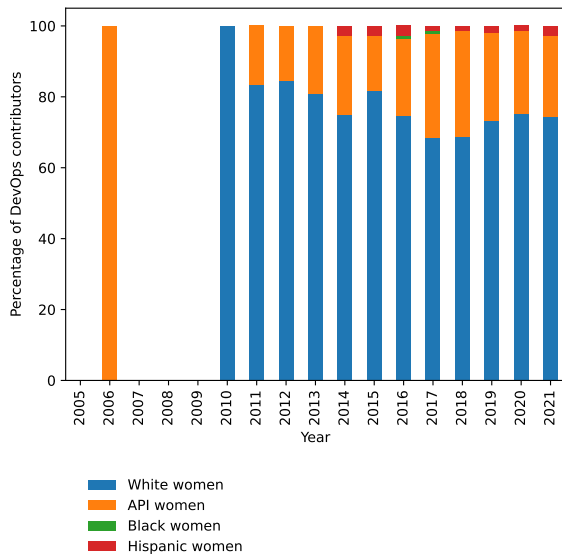
Table 6.5: Results of the statistical analysis of the percentages of the perceptible ethnicities of perceptibly women DevOps and non-DevOps contributors.

Perceptible Group	DevOps (Median)	Non-DevOps (Median)	Wilcoxon, One-Tailed, Paired ( $\alpha = 0.0023$ )	Effect Size (Cliff's $ \delta $ )
White women	100%	70%	$p = 5.85 \times 10^{-5} \ll \alpha$	0.221, small
API women	0%	25%	$p = 3.48 \times 10^{-4} < \alpha$	0.219, small
Hispanic women	0%	0%	$p = 3.07 \times 10^{-6} \ll \alpha$	0.193, small
Black women	0%	0%	$p = 0.18 > \alpha$	-

compared to non-DevOps, with a small effect size (Wilcoxon,  $p \ll \alpha = 0.0023$ , one-tailed, paired; Cliff's  $|\delta| = 0.221$ ). With respect to contributors who belong to the intersection of two minorities, perceptibly API women and Hispanic women contributors are more underrepresented among DevOps contributors compared to non-DevOps contributors, with a small effect size. Lastly, we do not observe a significant difference between the DevOps and non-DevOps contributors who are perceived as black women; however, we observe that a substantial proportion of projects do not contain such contributors. In particular, 34% of the projects do not have perceptibly black women among DevOps contributors, while 4% of the projects do not have any perceptibly black women among non-DevOps contributors.

**Observation 6(6): Over time, the perceptible ethnic diversity of perceptibly women DevOps and non-DevOps contributors is increasing. Perceptibly women contributors with perceptibly White names remain the majority for both DevOps and non-DevOps contributors.** Fig. 6.11 presents (a) the percentages of different perceptible ethnicities per year within DevOps contributors who are perceptibly women, and (b) the values of the corresponding diversity metrics per year for DevOps contributors who are perceptibly women. This figure shows that the number of perceptible ethnicity groups of DevOps contributors with perceptibly women names is increasing. For example, the percentage of perceptibly API women contributors has increased from 0% in 2010 to 22.9% in 2021, excluding the exception of 2006). This is further evident in Fig. 6.11b, as it shows an overall increase in diversity metrics with respect to the perceived ethnic diversity of perceptibly women DevOps contributors. For example, the ethnic evenness of DevOps contributors who are perceptibly women increased from zero in 2010 to 0.218 in 2021. Still, we observe that the improvement in evenness over the last years is not substantial; the average growth in evenness over the last ten years is 0.007. We observe similar trends for the Blau index and diffusion score as well.

For a subset of authors (7.4%), ethnic and gender-inferring tools could not identify both



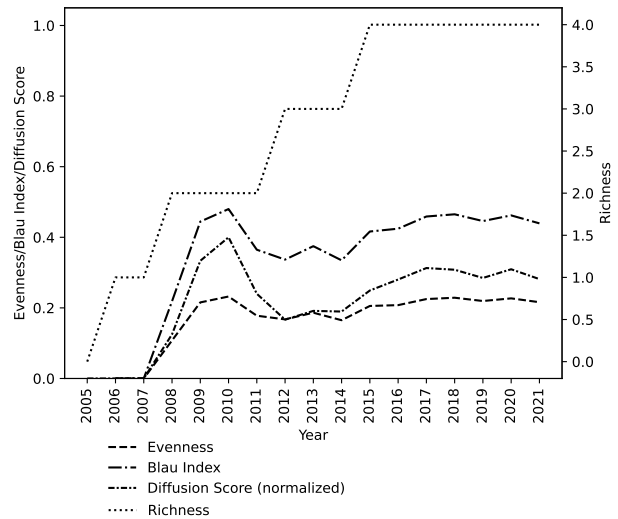
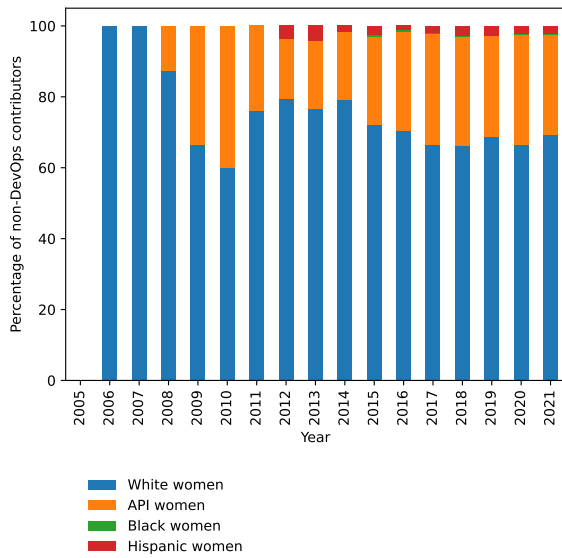
(a) Percentages of perceptible ethnicities of DevOps contributors who are perceptibly women.

(b) Values of diversity metrics over time: Both the left y-axis and the right y-axis present the metrics values. Note that the diffusion score is normalized.<sup>93</sup>

Figure 6.11: Change in the perceptible ethnic diversity of the DevOps contributors who are perceptibly women.

the perceptible ethnicity and the gender with the expected confidence level. Thus, for years 2005, 2007, 2008, and 2009 in Fig. 6.11a, we do not observe perceptibly women DevOps contributors with a perceptible ethnicity confidence level greater than 0.8. For 2006, we find an exception where only one perceptibly women DevOps contributor is available with a perceptible ethnicity confidence level greater than 0.8 in our dataset.

With respect to non-DevOps contributors, Fig. 6.12 shows that contributors who are perceived as women from different ethnicities follow a similar trend to DevOps contributors. In particular, the richness is increasing, yet the other metrics have remained almost constant in the last ten years; for example, the evenness ranges between 0.165–0.229, and the average growth is only 0.005.



(a) Percentages of perceptible ethnicities of non-DevOps contributors who are perceptibly women.

(b) Values of diversity metrics over time: Both the left y-axis and the right y-axis present the metrics values. Note that the diffusion score is normalized.<sup>93</sup>

Figure 6.12: Change in the perceptible ethnic diversity of the non-DevOps contributors who are perceptibly women.

Contributors who are perceived as White women are the majority of the women contributors, followed by contributors with perceptibly API, Hispanic, and Black names. We find statistical differences between the participation of contributors perceived as women in DevOps and non-DevOps (e.g., contributors who are perceptibly API and Hispanic women are more underrepresented in DevOps compared to non-DevOps). Additionally, it is worth noting that perceptibly black women are absent among DevOps contributors in 34% of the projects, while among non-DevOps contributors, this is only 4%. Over time, the richness of perceptible ethnic diversity of perceptibly women contributors tends to increase, yet the other diversity metrics remain low.

## 6.4 Threats to Validity

In this section, we describe the threats to the validity of this chapter. To support verifiability and replicability, we make our replication package publicly available.<sup>91</sup>

### 6.4.1 Construct Validity

We use the NAME-PRISM [182] tool to infer the perceptible ethnicity of contributors. Using NAME-PRISM to infer ethnicity has three main limitations. First, the ethnicity of an individual is a complex and multifaceted social construct that is not always easily identifiable [48], and the tool may misassign ethnicities to contributors. For example, contributors with Brazilian names but are not Hispanic (because their ancestors do not speak Spanish) may be perceived as Hispanic by the NAME-PRISM. Mixed-race contributors also may be perceived as belonging to a particular ethnicity, which may not necessarily align with the ethnicity that they identify with personally. Also, Black individuals may have names perceived as White [53], and using the perceived ethnicity may exclude individuals who identify as Black. However, note that in open-source communities, such as GitHub, one would only perceive the ethnicity of the other unless and otherwise it is revealed. Thus, our goal is to study the perceived diversity instead of the actual diversity. Moreover, as suggested by prior work [120], we only use contributors' names in which the perceptible ethnicity is inferred by the tool with a confidence level greater than 0.8 for better validity.

The second limitation of using NAME-PRISM is that the US government officially categorizes people with origins in Lebanon, Iran, Egypt, and other countries in the Middle East and North Africa (MENA) region as White.<sup>95</sup> According to Maghbouleh et al. [108], both non-MENA Whites and MENA individuals consider MENA-related traits as MENA rather than White. In addition, treating individuals from the MENA region as Whites does not necessarily correspond to the discrimination-related experiences of these contributors.<sup>96</sup> Because of this, considering individuals from the MENA region as White as per US Census Bureau may not accurately reflect the perceptible diversity of the DevOps and non-DevOps contributors that we study. However, to check the validity of our conclusions, we recompute our results by considering the perceptibly White contributors with perceptible MENA-related traits as a separate ethnic group. Since the contributor names available in our current dataset do not include the necessary information (e.g., nationality) to check

---

<sup>95</sup><https://www.documentcloud.org/documents/21218285-2020-census-state-pl-94-171-redistricting-summary-file-technical-documentationdocument/p232/a2082431>

<sup>96</sup><https://www.npr.org/2022/02/17/1079181478/us-census-middle-eastern-white-north-african-mena>

if a contributor is associated with a **MENA**-related trait or not, we use NAME-PRISM’s perceptible nationality-inferring API (which has an F-score of 0.795). We find our conclusions remain unchanged. An in-depth preview of our results corresponding to this new distinction of contributors who are perceived as from **MENA** countries is available in our Online Appendix E.<sup>91</sup>

The third limitation of using the NAME-PRISM is that the results obtained with respect to **US** Census Bureau’s classification may not be completely generalizable for contributors who are based outside **US** [5, 141]. To reflect on the validity of our study, we perform a new analysis to understand the impact of using NAME-PRISM for both **US**-based and non-**US**-based contributors. To categorize contributors into *US-based* and *Outside US*, we attempt to use the GitHub API to obtain the geographical locations, specifically the countries, of the contributors within our dataset. Because our current dataset does not include GitHub usernames, which are exclusively needed to retrieve locations via GitHub API,<sup>97</sup> we manually check for GitHub profiles of those contributors for locations. Note that NAME-PRISM’s perceptible nationality-inferring API that we used in our previous analysis on contributors perceptibly associated with **MENA**-related traits does not infer locations but rather nationalities; for example, it does not make the distinction between perceptible Whites from the **US** vs. the perceptible Whites from other countries. For our manual check, we obtain a significant sample of 500 contributors by searching on GitHub, and we check if there is a substantial difference in the observations we made with respect to the two distinct groups based on their locations: *US-based* and *Outside US*. Note that this sample size of 500 individuals exceeds the minimum recommended threshold (383) for achieving a 95% confidence level with a 5% margin of error when making inferences about the entire population of contributors within our dataset. We do not observe substantial changes in the observations concerning the two groups. For example, we find that, regardless of the location, perceptibly White contributors are the majority; contributors from other perceptible ethnicities are generally underrepresented. This is consistent with the observations we made in Section 6.3.1. We provide a detailed preview of our results corresponding to this new analysis in our Online Appendix J.<sup>91</sup> Furthermore, to facilitate future studies on diversity, we encourage researchers to train more sophisticated ethnicity-inferring tools with datasets spanning diverse populations.

We use gender-inferring tools (e.g. **Gender-guesser**) to infer the perceptible gender of contributors. Using such tools to infer gender has two main limitations. First, gender-inferring tools may not recognize the perceptible gender correctly as discussed in Section 6.2.2 (DC-5), e.g., rate of unrecognized names in Gender-guesser was 20.12% [142]. We tackle this problem by combining the outcomes from three gender-inferring tools to

---

<sup>97</sup><https://docs.github.com/en/free-pro-team@latest/rest/users/users?apiVersion=2022-11-28get-a-user>

take advantage of the strengths of all the tools as discussed in Section 6.2.2 (DC-5). We use the most effective combination, which we obtain by testing different combinations of the tools against the labelled dataset provided by Santamaría et al. [142]. The selected combination approach reduces the rate of unrecognized names to 5.15%. We remove these names with unrecognized genders from our analyses for better accuracy.

The second limitation of using gender-inferring tools is that gender-inferring tools have different classifications of gender given a name. To ensure consistency across all the tools used in this study, we follow the definition of gender as a binary classification (either men or women) as used in prior studies by Vasilescu et al. [166, 168, 167]. As mentioned in Section 6.2.2. (DC-5), we use three gender inferring tools: `Wiki-Gendersort`, `Gender-guesser`, and `GenderComputer`. The first tool, `Wiki-Gendersort`, classifies a name into one of the three gender groups: `male`, `female`, and `unisex`. We consider `female` category as women and `male` category as men; we consider the remaining category, `unisex`, to be unknown because we limit our discussion to a binary classification of gender [166, 168, 167]. The next tool, `Gender-guesser`, classifies a name into one of the five categories: `female`, `mostly female`, `androgynous`, `mostly male`, and `male`. We consider `female` and `mostly female` categories as women; similarly, we consider `male` and `mostly male` categories as men. We consider `Gender-guesser`'s remaining category (`androgynous`) as unknown, as similar to `Wiki-Gendersort`'s `unisex` category. The third tool, the `GenderComputer`, classifies gender into two main categories: `male` and `female`. Note that we cannot claim `Gender-guesser`'s `androgynous` category and `Wiki-Gendersort`'s `unisex` category to have the same meaning without evidence. We consider the `female` category inferred from `GenderComputer` to be women, and the `male` category to be men. Although we are aware that gender is a more complex issue than a binary variable [142], we believe that following the terminologies used in prior work mitigates the risk of misrepresenting genders. Future researchers may consider training more sophisticated gender-inferring tools with much larger datasets, taking the diverse representations of gender into consideration.

We use Vasilescu et al.'s `GitHub-alias-merging` approach [169] to track name changes of users, as discussed in Section 6.2.2 (DC-4). If a user changes the name to a completely different one but keeps using the same email address, we are able to unify the identities of such cases. However, if the user makes substantial changes to both their name and email address, we may miss such cases. In fact, it is infeasible to recognize such cases by human contributors as well since no ground truth data is available for users. Nonetheless, we only discuss the perceived diversity in this paper, and missing such extreme cases would not substantially impact our study's conclusions.

Lastly, in our study, we use four measures of diversity (richness [37], evenness [20], Blau index [149], and diffusion score<sup>92</sup>). Other indices, such as Gini index [61] and Theil

index [163], are not applicable to our study because they are not comparable across different teams when teams have different numbers of ethnicities or genders, as in our study.

### 6.4.2 Internal Validity

To identify DevOps files, we rely on filename and keyword conventions. Despite our best efforts, our automated classification script may still misclassify files. However, our initial results from a validation set (of 400 files) show that the agreement between a human labeller and the automated classification script is near-perfect (Cohen’s Kappa = 0.82).

Our preliminary analysis of the dataset shows that a subset of DevOps files are automatically generated and, hence, are not valid targets for our analyses. We filter out such files using keyword matching. Initial results from a validation set of 400 randomly selected files that are classified as DevOps show that the agreement between a human labeller and the automated classification script is near-perfect (i.e., Cohen’s Kappa of 0.81).

We considered DevOps contributors as the ones who made at least one change to DevOps files. A threshold of one DevOps file change may introduce selection bias to our definition of DevOps contributors. To address this, we recompute our results with two more thresholds: two file changes and ten file changes. We find our results still hold, indicating that our choice of the threshold does not include a substantial selection bias. A detailed description of the results of this extra analysis is available in our Online Appendices C and D.<sup>91</sup>

### 6.4.3 External Validity

We begin with a dataset of projects using CircleCI, and carefully select a meaningful sample of projects that are worthy of study. To do so, we filter out the projects that are immature and have little development history by applying filtration criteria used by prior studies [91]. This selection results in a dataset of 450 projects that adopt DevOps tools and technologies. Our dataset might be considered small when it is compared to the whole population of GitHub projects. To check the validity of our analyses beyond the projects that adopt CircleCI, we recompute the results of our research questions using another dataset that contains commits of projects that adopt GitHub Actions.<sup>8</sup> We find that the overall conclusions of the study remain unchanged for this new dataset of projects that adopt GitHub Actions. To access a complete preview of this analysis and its corresponding figures, we direct readers to our Appendix F.<sup>91</sup>

Lastly, while we study ethnic diversity in open-source projects, prior work shows that a majority of open-source contributors are based in North America and Europe [120]. In non-Western or company-specific settings, both the demographic makeup and the emphasis placed on different diversity dimensions may vary. For instance, in certain contexts, aspects such as neurodiversity and age diversity among DevOps developers may be more relevant and warrant further exploration in future studies.

## 6.5 Practical Implications

In this chapter, we investigate the perceived gender and ethnic diversity of DevOps and non-DevOps contributors to open-source projects by quantitatively analyzing 4,207,735 commits made by 110,336 contributors to 450 open-source projects in GitHub. Below, we discuss the implications of our findings.

**The lack of perceived diversity among DevOps contributors) is more prominent than that of the other developers.** Our findings indicate disparities in the perceived ethnic and gender representation among DevOps and non-DevOps contributors in the GitHub community. In particular, our observations 6(1) and 6(2) show that the group of contributors who contribute to DevOps artifacts tend to have less perceptible diversity than contributors to non-DevOps artifacts. Previous studies [92, 101] have discussed barriers to adopt DevOps, such as the difficulties in choosing appropriate tools from a diverse set of tools, which may contribute to the lack of perceptible diversity, which, in turn, can serve as an additional—albeit tacit—barrier to adoption. Potential reasons for the lack of diversity in DevOps have been discussed in various online reports as well. For instance, the report “Inspiring Women to Join the DevOps Movement”<sup>98</sup> discussed a lack of awareness among women developers about the availability of career paths such as DevOps. Other reports<sup>99,100</sup> also discussed the stereotypical perception that DevOps is man-dominated, which may discourage women from pursuing careers in this area. Concerns were raised about the gender pay gap<sup>101,102</sup> in DevOps, which may also contribute to the lack of diversity. We encourage future studies to direct efforts towards further understanding why DevOps contributions (e.g., contributing to CI configuration files) are less attractive to perceptibly non-Whites and women developers. We also encourage future work to extend

---

<sup>98</sup><https://www.pagerduty.com/blog/inspiring-women-to-join-the-devops-movement>

<sup>99</sup><https://www.cloudbees.com/blog/women-devops-tracy-ragan>

<sup>100</sup><https://peaplealent.com/the-gender-gap-in-technology>

<sup>101</sup><https://www.puppet.com/press/press-releases/puppets-seventh-annual-devops-salary-report>

<sup>102</sup><https://techmonitor.ai/leadership/workforce/devops-gender-pay-gap-critics-calling-change>

our results by examining how this pronounced lack of diversity among DevOps contributors impacts business practices. In particular, we suggest measuring variations in developer productivity and the quality of source code in projects with differing levels of diversity, using frameworks such as SPACE [52] and TRUCE [158], respectively.

**The perceived diversity of DevOps and non-DevOps contributors is slowly increasing over time, but there is still room for improvement.** Observations 6(3) and 6(4) show that perceptible diversity is increasing over time, but the improvement is not substantial. To bridge this gap, one particular effort is the “Women in DevOps” platform,<sup>68</sup> which was established in 2017 specifically to address the issue of gender imbalance in the DevOps industry. Furthermore, to improve diversity in Software teams in general, previous studies [173, 120, 132] have suggested strategies. For example, Wang et al. [173] have proposed to design a series of carefully crafted and empirically tested training courses that aim to reduce gender bias in both educational institutions and software development organizations. Prana et al. [132] emphasized the fact that current automatic tools, such as bug assignment tools [100, 88], make recommendations based on the similarity between developers (homophily), restricting the promotion of diversity. That said, to the best of our knowledge, it is yet to be investigated whether projects get the benefit of implementing such strategies. Future work could examine the impact of various strategies, such as mentorship programs [132, 173] and code of conduct amendments [132, 50], shedding light on specific interventions required to promote diversity and inclusion for developers contributing to different project activities (e.g., DevOps and non-DevOps).

**The lack of perceived diversity is amplified when considering the intersection of ethnicity and gender of DevOps and non-DevOps contributors.** Prior work [119, 120, 168, 18] that studied diversity had mainly considered independent analyses of ethnic and gender diversity. Considering the intersection of perceived ethnic and gender diversity is important to provide a richer understanding of the problems that individuals in this intersection encounter. For example, our observations 6(5) and 6(6) show that DevOps and non-DevOps contributors who are perceptibly Hispanic women and Black women are greatly underrepresented (median = 0%), while DevOps contributors who are perceived as API women and Hispanic women tend to be more underrepresented than non-DevOps contributors. We believe that the challenges faced by those who belong at the intersection of a minority ethnicity and minority gender may not be treated with a single overarching solution targeted towards one identity factor. Thus, we encourage future studies to explore those challenges and amplify the voices of developers situated at the intersection.

## 6.6 Chapter Summary

Among tacit inefficiencies related to CI, the lack of diversity plays an important role. While prior work has raised concerns about the lack of diversity in open-source communities, it remains unclear whether these diversity concerns extend to DevOps contributors—those who contribute to CI-related artifacts alongside other DevOps artifacts within projects.

In this chapter, we present an empirical study to analyze perceptible ethnic and gender diversity among DevOps contributors while grounding our analysis in a comparison with non-DevOps contributors. Below, we summarize the key highlights of this chapter.

- The lack of perceptible diversity is more pronounced among team members who contribute to DevOps artifacts (e.g., CI configuration files) than among others, calling for a deeper understanding of why DevOps work is less attractive to perceptible minorities.
- The perceived diversity among both DevOps and non-DevOps contributors has been gradually increasing over time; however, there remains considerable room for improvement.
- The lack of perceived diversity is amplified when considering the intersection of minority ethnicities and genders among DevOps and non-DevOps contributors, highlighting the need for targeted inclusion strategies to address compounded underrepresentation.

**Concluding Remarks.** In this thesis, we discussed tacit inefficiencies in CI, such as timeouts and updates to unused dependencies, as well as tacit barriers, including the lack of diversity among CI contributors. We also presented insights into opportunities for improvement. In the next and final chapter, we conclude the thesis and discuss potential future directions.

## **Part IV**

# **Final Remarks**

# Chapter 7

## Conclusions and Future Work

Continuous Integration (CI) is the heartbeat of a software project. It enables team members to validate change sets through automated cycles (i.e., CI builds). The events of importance, such as commits being submitted to a shared repository and creation of PRs, can automatically trigger CI builds. These CI builds orchestrate a series of tasks, including dependency retrieval, compilation, and testing to validate any changes and maintain code integrity [170].

While adoption of CI improves team productivity [155, 80] and software quality [170, 81, 156], these benefits come at a cost. In this thesis, we set out to understand the prevalence and characteristics of tacit inefficiencies and barriers in CI and provide strategies to mitigate them. We use historical data from CI pipelines and evaluate the following research hypothesis.

**Hypothesis.** Neglecting tacitly accrued inefficiencies and barriers in CI can have a substantial impact on both CI resources (i.e., build time) and the team members in a project who contribute to CI artifacts. Systematically identifying and characterizing these issues can inform the development of strategies that enhance the overall efficiency of CI pipelines.

To evaluate this hypothesis, we conduct empirical studies focusing on tacit inefficiencies in CI, such as timeouts occurring in the environments where CI builds are executed and wasteful CI builds triggered by updates to unused dependencies. We perform another empirical study that investigates tacit barriers in CI, such as the lack of contributor diversity.

## 7.1 Contributions and Findings

In this section we summarize the main contributions and findings of this thesis.

**Part II Tacit inefficiencies** stem from the environments where CI builds are executed and from the external dependencies declared in project. In Chapter 4, we examine the prevalence and characteristics of CI timeouts that are caused by inefficiencies in CI environments. Our findings suggest that project build history and timeout clusters can offer useful insights to proactively allocate resources and reduce CI waste. We also find that certain files are more prone to timeout builds than others and thus could help optimize resource allocation and potentially prevent timeout builds. In Chapter 5, we investigate inefficiencies in project dependencies, particularly focusing on updates to unused dependencies that trigger wasteful CI builds. This chapter also highlights the need for bot developers to take CI waste caused by unnecessary updates into account, as bots are the major culprits of triggering updates to unused dependencies. For project teams, we introduce an approach, DEP-SCIMITAR<sub>u</sub>, designed to minimize CI waste accrued from unused dependencies. We also encourage CI providers to recommend DEP-SCIMITAR<sub>u</sub> to project teams that excessively consume CI build time due to such wasteful builds.

**Part III Tacit barriers** are the challenges that stem from social, cultural, and organizational factors. With respect to such barriers, Chapter 6 presents an empirical study on the diversity and inclusion of developers who contribute to DevOps artifacts (e.g., CI configuration files) within projects, i.e., DevOps contributors. This study reveals that the perceptible diversity of DevOps contributors is significantly lower than that of other contributors in project teams. Although the perceived diversity of both DevOps and non-DevOps contributors has gradually increased over time, there remains substantial room for improvement. Moreover, the lack of perceived diversity is amplified when examining the intersection of ethnicity and gender among contributors, calling for targeted inclusion strategies.

Overall, this thesis contributes to a broader understanding of inefficiencies and barriers that are not explicitly recognized as specific to CI, as well as practical insights for CI stakeholders and researchers aiming to optimize CI.

## 7.2 Future Work

This thesis lays the groundwork for promising future directions, which we detail below.

### 7.2.1 Develop approaches for predicting timeout builds

Our observations in Chapter 4 indicate that our model demonstrates strong discriminatory power, is well-calibrated, and exhibits high stability in explaining CI timeouts. Although our focus is on using statistical models to characterize and understand timeout builds rather than predict future occurrences, we encourage future research to expand upon our findings on the key features that explain CI timeouts. These insights, along with related work on build failure prediction, can inform the development of more robust models for predicting timeout builds. For instance, previous studies (e.g., Chen et al. [25]) have used machine-learning models to predict build failures; our results suggest that similar approaches could be adapted to predict CI timeouts.

### 7.2.2 Extend the scope of the impact of unused dependencies beyond CI build time

While our observations in Chapter 5 highlight the impact of unused dependencies on CI, future research may investigate their broader implications on software development and maintenance. For instance, this thesis finds that unused dependencies are frequently updated to newer versions as they become available. This may reflect organizational concerns about the security risks associated with outdated dependencies [98], regardless of actual usage. Such practices contribute to development overhead, indicating that further investigation into this area could offer valuable insights to mitigate development overhead.

### 7.2.3 Mitigate inefficiencies in CI due to smells in CI configurations

In this thesis, we discussed two sources of tacit inefficiencies in CI: CI environment and project dependencies. However, other sources of tacit inefficiencies also exist, such as those arising from CI configuration files. CI configurations are susceptible to anti-patterns, a.k.a CI smells [58, 171, 184, 21]. While prior studies partially went in this direction [186], mitigation strategies for a variety of smells are yet to be addressed. For example, the

use of overly long scripts that perform redundant tasks [148] and incorrect parallelization settings [58] are examples of CI smells that need further investigation. These CI smells, often subtle and unnoticed, can degrade the performance and effectiveness of CI pipelines [58, 171, 184, 21]. Thus, we encourage future studies to quantify the impact of these CI smells and propose strategies to mitigate them.

#### **7.2.4 Explore new business models for CI providers and project maintainers that benefit long-term sustainability**

In Part II (i.e., tacit inefficiencies), we discuss that both CI providers and project maintainers may bear the cost of inefficient CI usage. We encourage future work to explore a new business model that encourages collaboration between CI providers and project teams to improve the sustainability and efficiency of CI pipelines. For instance, while CI providers often do not disclose internal data related to resource usage, they could expose actionable insights to project teams through subscriptions, e.g., allowing them to allocate extra time for timeout-prone builds to complete in exchange for a reasonable fee. Such a model would help reduce redundant executions, alleviate infrastructure strain, and create mutual benefits for both CI providers and their customers.

#### **7.2.5 Explore the challenges of DevOps contributors situated at the intersection of minority groups**

Our observations in Chapter 6 show that DevOps contributors who are perceptible as Asian/Pacific Islander (API) women and Hispanic women tend to be more underrepresented than non-DevOps contributors. However, prior work [119, 120, 168, 18] that studied diversity has independently analyzed challenges faced by developers from minority ethnicities and genders and proposed solutions to improve diversity and inclusion. We believe that the challenges faced by those who are perceptible at the intersection of a minority ethnicity and minority gender may not be treated with a single overarching solution targeted towards one identity factor. Thus, we encourage future studies to explore the challenges faced by DevOps contributors who are situated at the intersection of multiple minority traits.

### **7.2.6 Explore tacit barriers faced by DevOps contributors beyond diversity and inclusion**

While this thesis focuses on diversity as a key tacit barrier in CI, there are other barriers that are yet to be explored, such as the personal well-being of DevOps contributors [110] and the cognitive workload of DevOps tasks [74]. For example, elements such as lack of motivation, distractions, fear of making mistakes, and burnout [130] can negatively impact productivity, collaboration, and the overall efficiency of CI pipelines and other DevOps tasks. Future work may aim to quantify the effects of these human-centered challenges and propose strategies to mitigate their negative impacts—both on developers and on software development. Such efforts could help foster a healthier work environment through improved workload management, mental health support, and the promotion of a more sustainable work culture.

# References

- [1] Periodic table of devops. <https://digital.ai/periodic-table-of-devops-tools>.
- [2] Rabe Abdalkareem, Suhaib Mujahid, and Emad Shihab. A machine learning approach to improve the detection of ci skip commits. *Transactions on Software Engineering*, 2020.
- [3] Rabe Abdalkareem, Suhaib Mujahid, Emad Shihab, and Juergen Rilling. Which commits can be ci skipped? *Transactions on Software Engineering*, 47, 2019.
- [4] Amal Akli, Guillaume Haben, Sarra Habchi, Mike Papadakis, and Yves Le Traon. Flakycat: Predicting flaky tests categories using few-shot learning. In *2023 IEEE/ACM International Conference on Automation of Software Test (AST)*, pages 140–151. IEEE, 2023.
- [5] Khaled Albusays, Pernille Bjorn, Laura Dabbish, Denae Ford, Emerson Murphy-Hill, Alexander Serebrenik, and Margaret-Anne Storey. The diversity crisis in software development. *Software*, 38, 2021.
- [6] Mahmoud Alfadel, Diego Elias Costa, and Emad Shihab. Empirical analysis of security vulnerabilities in python packages. *Empirical Software Engineering*, 28, 2023.
- [7] Mahmoud Alfadel, Diego Elias Costa, Emad Shihab, and Bram Adams. On the discoverability of npm vulnerabilities in node.js projects. *ACM Transactions on Software Engineering and Methodology*, 32(4):1–27, 2023.
- [8] Mahmoud Alfadel, Diego Elias Costa, Emad Shihab, and Mouafak Mkhallalati. On the use of dependabot security pull requests. In *18th International Conference on Mining Software Repositories (MSR)*, 2021.

- [9] Abdullateef Oluwagbemiga Balogun, Shuib Basri, Jadid Abdulkadir Said, Victor Ebenezer Adeyemo, Abdullahi Abubakar Imam, and Amos Orenyi Bajeh. *Software defect prediction: analysis of class imbalance and performance stability*. School of Engineering, Taylor’s University, 2019.
- [10] Soon K Bang, Sam Chung, Young Choh, and Marc Dupuis. A grounded theory analysis of modern web applications: knowledge, skills, and abilities for devops. In *Proceedings of the 2nd annual conference on Research in information technology*, 2013.
- [11] Len Bass, Ingo Weber, and Liming Zhu. *DevOps: A software architect’s perspective*. 2015.
- [12] Gabriele Bavota, Gerardo Canfora, Massimiliano Di Penta, Rocco Oliveto, and Sebastiano Panichella. How the apache community upgrades dependencies: an evolutionary study. *Empirical Software Engineering*, 20, 2015.
- [13] Moritz Beller, Georgios Gousios, and Andy Zaidman. Oops, my tests broke the build: An explorative analysis of travis ci with github. In *14th International conference on mining software repositories (MSR)*, 2017.
- [14] Arka Bhattacharya. *Impact of continuous integration on software quality and productivity*. PhD thesis, The Ohio State University, 2014.
- [15] Chris Bogart, Christian Kästner, James Herbsleb, and Ferdian Thung. When and how to make breaking changes: Policies and practices in 18 open source software ecosystems. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 30(4):1–56, 2021.
- [16] Christopher Bogart, Christian Kästner, and James Herbsleb. When it breaks, it breaks: How ecosystem developers reason about the stability of dependencies. In *International Conference on Automated Software Engineering Workshop*, 2015.
- [17] Carlo Bonferroni. Teoria statistica delle classi e calcolo delle probabilita. *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commerciali di Firenze*, 8, 1936.
- [18] Amiangshu Bosu and Kazi Zakia Sultana. Diversity and inclusion in open source software (oss) projects: Where do we stand? In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*, 2019.

- [19] Glenn W Brier. *Verification of forecasts expressed in terms of probability*, volume 78. American Meteorological Society, 1950.
- [20] Leon Brillouin. Maxwell’s demon cannot operate: Information and entropy. i. *Journal of Applied Physics*, 22, 1951.
- [21] William J Brown, Hays W “Skip” McCormick III, and Scott H Thomas. *AntiPatterns and patterns in software configuration management*. John Wiley & Sons, Inc., 1999.
- [22] Tanja Buch, Moritz Meister, and Annekatrin Niebuhr. Ethnic diversity and segregation in german cities. *Cities*, 115, 2021.
- [23] Gemma Catolino, Fabio Palomba, Damian A Tamburri, Alexander Serebrenik, and Filomena Ferrucci. Gender diversity and women in software teams: How do they affect community smells? In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Society*, 2019.
- [24] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16, 2002.
- [25] Bihuan Chen, Linlin Chen, Chen Zhang, and Xin Peng. Buildfast: History-aware build outcome prediction for fast feedback and reduced cost in continuous integration. In *Proceedings of the 35th International Conference on Automated Software Engineering*, 2020.
- [26] Gerry Gerard Claps, Richard Berntsson Svensson, and Aybüke Aurum. On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software technology*, 57:21–31, 2015.
- [27] Filipe Roseiro Cogo, Gustavo A Oliva, and Ahmed E Hassan. An empirical study of dependency downgrades in the npm ecosystem. *Transactions on Software Engineering*, 47, 2019.
- [28] Jacob Cohen. Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. *Psychological Bulletin*, 70, 1968.
- [29] Russ Cox. Surviving software dependencies. *Communications of the ACM*, 62(9):36–43, 2019.

- [30] Daniel Cukier. Devops patterns to scale web applications using cloud services. In *Proceedings of the companion publication for conference on Systems, programming, & applications: software for humanity*, 2013.
- [31] Ozren Dabic, Emad Aghajani, and Gabriele Bavota. Sampling projects in github for msr studies. In *18th International Conference on Mining Software Repositories (MSR)*, 2021.
- [32] Jennifer L Davidson, Rithika Naik, Umme Ayda Mannan, Amir Azarbakht, and Carlos Jensen. On older adults in free/open source software: reflections of contributors and community leaders. In *Proceedings of the Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2014.
- [33] Adam Debbiche, Mikael Dienér, and Richard Berntsson Svensson. Challenges when adopting continuous integration: A case study. In *Product-Focused Software Process Improvement: 15th International Conference, PROFES 2014, Helsinki, Finland, December 10-12, 2014. Proceedings 15*, pages 17–32. Springer, 2014.
- [34] Derek DeBellis, Kevin M. Storer, Amanda Lewis, Benjamin Good, Daniella Villalba, Eric Maxwell, Kim Castillo, Michelle Irvine, and Nathen Harvey. The accelerate state of devops report. 2024.
- [35] Alexandre Decan, Tom Mens, and Eleni Constantinou. On the impact of security vulnerabilities in the npm package dependency network. In *Proceedings of the 15th international conference on mining software repositories*, 2018.
- [36] Alexandre Decan, Tom Mens, and Hassan Onori Delicheh. On the outdatedness of workflows in the github actions ecosystem. *Journal of Systems and Software*, 2023.
- [37] Theodore M DeJong. A comparison of three diversity indices based on their components of richness and evenness. *Oikos*, 1975.
- [38] Tapajit Dey, Sara Mousavi, Eduardo Ponce, Tanner Fry, Bogdan Vasilescu, Anna Filippova, and Audris Mockus. Detecting and characterizing bots that commit code. In *Proceedings of the 17th international conference on mining software repositories*, 2020.
- [39] Elisa Diel, Sabrina Marczak, and Daniela S Cruzes. Communication challenges and strategies in distributed devops. In *2016 IEEE 11th International Conference on Global Software Engineering (ICGSE)*, pages 24–28. IEEE, 2016.

- [40] Thomas Durieux, Claire Le Goues, Michael Hilton, and Rui Abreu. Empirical study of restarted and flaky builds on travis ci. In *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020.
- [41] Paul M Duvall, Steve Matyas, and Andrew Glover. *Continuous integration: improving software quality and reducing risk*. 2007.
- [42] Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. *Devops. Software*, 2016.
- [43] Bradley Efron. How biased is the apparent error rate of a prediction rule? *Journal of the American statistical Association*, 81, 1986.
- [44] Joseph G Eisenhauer. *Degrees of Freedom in Statistical Inference*. Springer Berlin Heidelberg, 2011.
- [45] Holly Else, Jeffrey M Perkel, et al. The giant plan to track diversity in research journals. *Nature*, 602, 2022.
- [46] Andreas Erlandsson and Hannes Lantz. Improving feedback loop by two-step continuous integration.
- [47] Hamed Esfahani, Jonas Fietz, Qi Ke, Alexei Kolomiets, Erica Lan, Erik Mavrinac, Wolfram Schulte, Newton Sanches, and Srikanth Kandula. Cloudbuild: Microsoft’s distributed and caching build service. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 11–20, 2016.
- [48] James D Fearon and David D Laitin. Violence and the social construction of ethnic identity. *International organization*, 54, 2000.
- [49] Wagner Felidré, Leonardo Furtado, Daniel A Da Costa, Bruno Cartaxo, and Gustavo Pinto. Continuous integration theater. In *International Symposium on Empirical Software Engineering and Measurement*, 2019.
- [50] Denae Ford, Reed Milewicz, and Alexander Serebrenik. How remote work can foster a more inclusive environment for transgender developers. In *Proceedings of the 2nd International Workshop on Gender Equality in Software Engineering*, 2019.
- [51] Denae Ford, Justin Smith, Philip J Guo, and Chris Parnin. Paradise unplugged: Identifying barriers for female participation on stack overflow. In *Proceedings of the 24th SIGSOFT International symposium on foundations of software engineering*, 2016.

- [52] Nicole Forsgren, Margaret-Anne Storey, Chandra Maddila, Thomas Zimmermann, Brian Houck, and Jenna Butler. The space of developer productivity: There’s more to it than you think. *Queue*, 19(1):20–48, 2021.
- [53] Roland G Fryer Jr and Steven D Levitt. The causes and consequences of distinctively black names. *The Quarterly Journal of Economics*, 119, 2004.
- [54] Keheliya Gallaba. *Improving the Robustness and Efficiency of Continuous Integration and Deployment*. PhD thesis, McGill University, 3480 Rue University, Montréal, QC, Canada, November 2021.
- [55] Keheliya Gallaba, John Ewart, Yves Junqueira, and Shane McIntosh. Accelerating continuous integration by caching environments and inferring dependencies. *Transactions on Software Engineering*, 2020.
- [56] Keheliya Gallaba, Maxime Lamothe, and Shane McIntosh. Lessons from Eight Years of Operational Data from a Continuous Integration Service: An Exploratory Case Study of CircleCI. In *Proc. of the International Conference on Software Engineering*, 2022.
- [57] Keheliya Gallaba, Christian Macho, Martin Pinzger, and Shane McIntosh. Noise and heterogeneity in historical build data: an empirical study of travis ci. In *Proceedings of the 33rd International Conference on Automated Software Engineering*, 2018.
- [58] Keheliya Gallaba and Shane McIntosh. Use and misuse of continuous integration features: An empirical study of projects that (mis) use travis ci. *IEEE Transactions on Software Engineering*, 46(1):33–50, 2018.
- [59] Taher Ahmed Ghaleb, Daniel Alencar Da Costa, and Ying Zou. An empirical study of the long duration of continuous integration builds. *Empirical Software Engineering*, 24, 2019.
- [60] Taher Ahmed Ghaleb, Daniel Alencar Da Costa, Ying Zou, and Ahmed E Hassan. Studying the impact of noises in build breakage data. *IEEE Transactions on Software Engineering*, 47, 2019.
- [61] Corrado Gini. On the measure of concentration with special reference to income and statistics. *Colorado College Publication, General Series*, 208, 1936.
- [62] Milos Gligoric, Lamyaa Eloussi, and Darko Marinov. Practical regression test selection with dynamic file dependencies. In *Proceedings of the International Symposium on Software Testing and Analysis*, 2015.

- [63] Mehdi Golzadeh, Alexandre Decan, and Tom Mens. On the rise and fall of ci services in github. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2022.
- [64] Georgios Gousios, Martin Pinzger, and Arie van Deursen. An exploratory study of the pull-based software development model. In *Proceedings of the 36th international conference on software engineering*, 2014.
- [65] Hannah Green. Disparity in discrimination: A study on the experience of minority women in the workplace. 2017.
- [66] Lucas Gren. On gender, ethnicity, and culture in empirical software engineering research. In *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering*, 2018.
- [67] Dale Griffin, Kai Li, and Ting Xu. Board gender diversity and corporate innovation: International evidence. *Journal of Financial and Quantitative Analysis*, 56, 2021.
- [68] Sarra Habchi, Guillaume Haben, Jeongju Sohn, Adriano Franci, Mike Papadakis, Maxime Cordy, and Yves Le Traon. What made this test flake? pinpointing classes responsible for test flakiness. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 352–363. IEEE, 2022.
- [69] James A Hanley and Barbara J McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143, 1982.
- [70] Niels Richard Hansen and Alexander Sokol. Degrees of freedom for nonlinear least squares estimation. *arXiv preprint arXiv:1402.2997*, 2014.
- [71] Frank E Harrell et al. *Regression modeling strategies: with applications to linear models, logistic regression, and survival analysis*, volume 608. 2001.
- [72] Frank E Harrell Jr, Kerry L Lee, Robert M Califf, David B Pryor, and Robert A Rosati. Regression modelling strategies for improved prognostic prediction. *Statistics in medicine*, 3, 1984.
- [73] Frank E Harrell Jr, Kerry L Lee, David B Matchar, and Thomas A Reichert. Regression models for prognostic prediction: advantages, problems, and suggested solutions. *Cancer treatment reports*, 69, 1985.

- [74] Sandra G Hart and Lowell E Staveland. Development of nasa-tlx (task load index): Results of empirical and theoretical research. In *Advances in psychology*, volume 52, pages 139–183. Elsevier, 1988.
- [75] Tawfiq Hasanin and Taghi Khoshgoftaar. The effects of random undersampling with simulated class imbalance for big data. In *international conference on information reuse and integration (IRI)*, 2018.
- [76] Foyzul Hassan and Xiaoyin Wang. Change-aware build prediction model for stall avoidance in continuous integration. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*, 2017.
- [77] Haibo He and Eduardo A Garcia. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21, 2009.
- [78] Runzhi He, Hao He, Yuxia Zhang, and Minghui Zhou. Automating dependency updates in practice: An exploratory study on github dependabot. *Transactions on Software Engineering*, 2023.
- [79] Kim Herzig, Michaela Greiler, Jacek Czerwonka, and Brendan Murphy. The art of testing less without sacrificing quality. In *Proceedings of the 37th IEEE International Conference on Software Engineering*, volume 1, 2015.
- [80] Michael Hilton, Nicholas Nelson, Danny Dig, Timothy Tunnell, Darko Marinov, et al. Continuous integration (ci) needs and wishes for developers of proprietary code. 2016.
- [81] Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. Usage, costs, and benefits of continuous integration in open-source projects. In *Proceedings of the 31st international conference on automated software engineering*, 2016.
- [82] Jez Humble and Joanne Molesky. Why enterprises must adopt devops to enable continuous delivery. *Cutter IT Journal*, 24, 2011.
- [83] Abbas Javan Jafari, Diego Elias Costa, Rabe Abdalkareem, Emad Shihab, and Nikolaos Tsantalis. Dependency smells in javascript projects. *Transactions on Software Engineering*, 48, 2021.
- [84] Ciera Jaspan, Matthew Jorde, Andrea Knight, Caitlin Sadowski, Edward K Smith, Collin Winter, and Emerson Murphy-Hill. Advantages and disadvantages of a monolithic repository: a case study at google. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, pages 225–234, 2018.

- [85] Xianhao Jin and Francisco Servant. A cost-efficient approach to building in continuous integration. In *Proceedings of the 42nd International Conference on Software Engineering*, 2020.
- [86] Xianhao Jin and Francisco Servant. Which builds are really safe to skip? maximizing failure observation for build selection in continuous integration. *Journal of Systems and Software*, 188, 2022.
- [87] Xianhao Jin and Francisco Servant. Hybridcisave: A combined build and test selection approach in continuous integration. *ACM Transactions on Software Engineering and Methodology*, 32(4):1–39, 2023.
- [88] Leif Jonsson, Markus Borg, David Broman, Kristian Sandahl, Sigrid Eldh, and Per Runeson. Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts. *Empirical Software Engineering*, 21, 2016.
- [89] Md Mahir Asef Kabir, Ying Wang, Danfeng Yao, and Na Meng. How do developers follow security-relevant best practices when using npm packages? In *Secure Development Conference (SecDev)*, 2022.
- [90] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. The promises and perils of mining GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014.
- [91] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. An in-depth study of the promises and perils of mining github. *Empirical Software Engineering*, 2016.
- [92] Muhammad Shoaib Khan, Abudul Wahid Khan, Faheem Khan, Muhammad Adnan Khan, and Taeg Keun Whangbo. Critical challenges to adopt devops culture in software organizations: a systematic review. 2022.
- [93] Gene Kim, Jez Humble, Patrick Debois, John Willis, and Nicole Forsgren. *The DevOps handbook: How to create world-class agility, reliability, & security in technology organizations*. It Revolution, 2021.
- [94] Ajiromola Kola-Olawuyi, Nimmi Rashinika Weeraddana, and Meiyappan Nagappan. The impact of code ownership of devops artefacts on the outcome of devops ci builds. In *Proceedings of the 21st International Conference on Mining Software Repositories*, pages 543–555, 2024.

- [95] Raula Gaikovina Kula, Daniel M German, Ali Ouni, Takashi Ishio, and Katsuro Inoue. Do developers update their library dependencies? an empirical study on the impact of security advisories on library migration. *Empirical Software Engineering*, 23, 2018.
- [96] Manish Kumar. The design and implementation of automated deployment pipelines for amazon web services: Gitops practices in the context of ci/cd pipelines using gitlab and infrastructure as code, 2024.
- [97] J Richard Landis and Gary G Koch. The measurement of observer agreement for categorical data. *biometrics*, 1977.
- [98] Jasmine Latendresse, Suhaib Mujahid, Diego Elias Costa, and Emad Shihab. Not all dependencies are equal: An empirical study on production dependencies in npm. In *37th International Conference on Automated Software Engineering*, 2022.
- [99] Matthew Ryan Lavery, Parul Acharya, Stephen A Sivo, and Lihua Xu. Number of predictors and multicollinearity: What are their effects on error and bias in regression? *Communications in Statistics-Simulation and Computation*, 48, 2019.
- [100] Seonah Lee, Rongxin Wu, Shing-Chi Cheung, and Sungwon Kang. Automatic detection and update suggestion for outdated api names in documentation. *Transactions on Software Engineering*, 47, 2019.
- [101] Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. A survey of devops concepts and challenges. *Computing Surveys*, 52, 2019.
- [102] Marko Leppänen, Simo Mäkinen, Max Pagels, Veli-Pekka Eloranta, Juha Itkonen, Mika V Mäntylä, and Tomi Männistö. The highways and country roads to continuous deployment. *Ieee software*, 32(2):64–72, 2015.
- [103] Jingyue Li, Nils B Moe, and Tore Dybå. Transition from a plan-driven process to scrum: a longitudinal case study on software quality. In *Proceedings of the 2010 ACM-IEEE international symposium on empirical software engineering and measurement*, pages 1–10, 2010.
- [104] Yuxin Liu, Deepika Tiwari, Cristian Bogdan, and Benoit Baudry. An empirical study of bloated dependencies in commonjs packages. *arXiv preprint arXiv:2405.17939*, 2024.

- [105] Ruth W Macarthy and Julian M Bass. An empirical taxonomy of devops in practice. In *2020 46th euromicro conference on software engineering and advanced applications (seaa)*, pages 221–228. IEEE, 2020.
- [106] Mateusz Machalica, Alex Samylnik, Meredith Porth, and Satish Chandra. Predictive test selection. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*, 2019.
- [107] Christian Macho, Shane McIntosh, and Martin Pinzger. Predicting build co-changes with source code change and commit categories. In *2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER)*, volume 1, pages 541–551. IEEE, 2016.
- [108] Neda Maghbouleh, Ariela Schachter, and René D Flores. Middle eastern and north african americans may not be perceived, nor perceive themselves, to be white. *Proceedings of the National Academy of Sciences*, 119, 2022.
- [109] Rungroj Maipradit, Dong Wang, Patanamon Thongtanunam, Raula Gaikovina Kula, Yasutaka Kamei, and Shane McIntosh. Repeated Builds During Code Review: An Empirical Study of the OpenStack Community. In *Proc. of the International Conference on Automated Software Engineering*, 2023.
- [110] Gerald Matthews, Lucy Joyner, Kirby Gilliland, Sian Campbell, Shona Falconer, Jane Huggins, et al. Validation of a comprehensive stress state questionnaire: Towards a state big three. *Personality psychology in Europe*, 7:335–350, 1999.
- [111] Shane McIntosh, Bram Adams, Meiyappan Nagappan, and Ahmed E Hassan. Mining co-change information to understand when build changes are necessary. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 241–250. IEEE, 2014.
- [112] Shane McIntosh, Bram Adams, Meiyappan Nagappan, and Ahmed E Hassan. Identifying and understanding header file hotspots in c/c++ build processes. *Automated Software Engineering*, 23:619–647, 2016.
- [113] Shane McIntosh and Yasutaka Kamei. Are Fix-Inducing Changes a Moving Target? A Longitudinal Case Study of Just-In-Time Defect Prediction. *Transactions on Software Engineering*, 44, 2018.

- [114] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering*, 21, 2016.
- [115] Samim Mirhosseini and Chris Parnin. Can automated pull requests encourage software developers to upgrade out-of-date dependencies? In *2017 32nd IEEE/ACM international conference on automated software engineering (ASE)*, 2017.
- [116] Kamla Modi, Judy Schoenberg, and Kimberlee Salmond. Generation stem: What girls say about science, technology, engineering, and math. *A Report from the Girl Scout Research Institute. New York, NY: Girl Scouts of the USA*, 2012.
- [117] Hamid Mohayjeji, Andrei Agaronian, Eleni Constantinou, Nicola Zannone, and Alexander Serebrenik. Investigating the resolution of vulnerable dependencies with dependabot security updates. In *20th International Conference on Mining Software Repositories (MSR)*, 2023.
- [118] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. Curating github for engineered software projects. *Empirical Software Engineering*, 2017.
- [119] Reza Nadri, Gema Rodriguez-Perez, and Meiyappan Nagappan. Insights into non-merged pull requests in github: Is there evidence of bias based on perceptible race? *Software*, 38, 2021.
- [120] Reza Nadri, Gema Rodríguez-Pérez, and Meiyappan Nagappan. On the relationship between the developer’s perceptible race and ethnicity and the evaluation of contributions in oss. *Transactions on Software Engineering*, 48, 2021.
- [121] Thiago Nicolini, Andre Hora, and Eduardo Figueiredo. On the usage of new javascript features through transpilers: The babel case. *IEEE Software*, 2023.
- [122] Doriane Olewicki, Mathieu Nayrolles, and Bram Adams. Towards language-independent brown build detection. In *Proceedings of the 44th International Conference on Software Engineering*, pages 2177–2188, 2022.
- [123] Helena Holmström Olsson, Hiva Alahyari, and Jan Bosch. Climbing the” stairway to heaven”—a multiple-case study exploring barriers in the transition from agile development towards continuous deployment of software. In *2012 38th euromicro conference on software engineering and advanced applications*, pages 392–399. IEEE, 2012.

- [124] Marco Ortu, Giuseppe Destefanis, Steve Counsell, Stephen Swift, Roberto Tonelli, and Michele Marchesi. How diverse is your team? investigating gender and nationality diversity in github teams. *Journal of Software Engineering Research and Development*, 5, 2017.
- [125] Cong Pan and Michael Pradel. Continuous test suite failure prediction. In *Proceedings of the 30th SIGSOFT International Symposium on Software Testing and Analysis*, 2021.
- [126] Pierpaolo Parrotta, Dario Pozzoli, and Davide Sala. Ethnic diversity and firms' export behavior. *European Economic Review*, 89, 2016.
- [127] Ivan Pashchenko, Henrik Plate, Serena Elisa Ponta, Antonino Sabetta, and Fabio Massacci. Vulnerable open source dependencies: Counting those that matter. In *Proceedings of the 12th International Symposium on Empirical Software Engineering and Measurement*, 2018.
- [128] Ivan Pashchenko, Duc-Ly Vu, and Fabio Massacci. A qualitative study of dependency management and its security implications. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, 2020.
- [129] Karl Pearson. Contributions to the mathematical theory of evolution. *Philosophical Transactions of the Royal Society of London. A*, 185, 1894.
- [130] Juanjo Pérez-Sánchez, Saima Rafi, Juan Manuel Carrillo de Gea, Joaquín Nicolás Ros, and José Luis Fernández Alemán. A theory on human factors in devops adoption. *Computer Standards & Interfaces*, page 103907, 2024.
- [131] Shaun Phillips, Thomas Zimmermann, and Christian Bird. Understanding and improving software build teams. In *Proceedings of the 36th international conference on software engineering*, pages 735–744, 2014.
- [132] Gede Artha Azriadi Prana, Denae Ford, Ayushi Rastogi, David Lo, Rahul Purandare, and Nachiappan Nagappan. Including everyone, everywhere: Understanding opportunities and challenges of geographic gender-inclusion in oss. *Transactions on Software Engineering*, 48, 2021.
- [133] Daniel Preoțiuc-Pietro and Lyle Ungar. User-level race and ethnicity predictors from twitter text. In *Proceedings of the 27th international conference on computational linguistics*, 2018.

- [134] Joseph Prusa, Taghi M Khoshgoftaar, David J Dittman, and Amri Napolitano. Using random undersampling to alleviate class imbalance on tweet sentiment data. In *2015 IEEE international conference on information reuse and integration*, 2015.
- [135] Huilian Sophie Qiu, Alexander Nolte, Anita Brown, Alexander Serebrenik, and Bogdan Vasilescu. Going farther together: The impact of social capital on sustained participation in open source. In *Proceedings of the 41st international conference on software engineering*, 2019.
- [136] Jon NK Rao and Alastair J Scott. The analysis of categorical data from complex sample surveys: chi-squared tests for goodness of fit and independence in two-way tables. *Journal of the American statistical association*, 76(374):221–230, 1981.
- [137] Monique Ross, Zahra Hazari, Gerhard Sonnert, and Philip Sadler. The intersection of being black and being a woman: Examining the effect of social computing relationships on computer science career choice. *Transactions on Computing Education*, 20, 2020.
- [138] Islem Saidani, Ali Ouni, Moataz Chouchen, and Mohamed Wiem Mkaouer. Predicting continuous integration build failures using evolutionary search. *Information and Software Technology*, 128, 2020.
- [139] Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PLoS one*, 10, 2015.
- [140] Mary Sánchez-Gordón and Ricardo Colomo-Palacios. Characterizing devops culture: a systematic literature review. In *Proceedings of the International Conference on Software Process Improvement and Capability Determination*, 2018.
- [141] Livio Sansone. *Blackness without ethnicity: Constructing race in Brazil*. 2003.
- [142] Lucía Santamaría and Helena Mihaljević. Comparison and benchmark of name-to-gender inference services. *PeerJ Computer Science*, 4, 2018.
- [143] Ville Satopaa, Jeannie Albrecht, David Irwin, and Barath Raghavan. Finding a” kneedle” in a haystack: Detecting knee points in system behavior. In *Proceedings of the 31st international conference on distributed computing systems workshops*, 2011.
- [144] Carolyn B. Seaman. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on software engineering*, 25(4):557–572, 1999.

- [145] Paul Sebo. Performance of gender detection tools: a comparative study of name-to-gender inference services. *Journal of the Medical Library Association: JMLA*, 109, 2021.
- [146] Mali Senapathi, Jim Buchan, and Hady Osman. Devops capabilities, practices, and challenges: Insights from a case study. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, pages 57–67, 2018.
- [147] August Shi, Wing Lam, Reed Oei, Tao Xie, and Darko Marinov. ifixflakies: A framework for automatically fixing order-dependent flaky tests. In *Proceedings of the 27th Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019.
- [148] Ruben Blencio Tavares Silva and Carla IM Bezerra. Analyzing continuous integration bad practices in closed-source projects: An initial study. In *Proceedings of the XXXIV Brazilian Symposium on Software Engineering*, pages 642–647, 2020.
- [149] Edward H Simpson. Measurement of diversity. *nature*, 163, 1949.
- [150] Dustin Smith, Daniella Villalba, Michelle Irvine, Dave Stanke, and Nathen Harvey. The accelerate state of devops report. 2019.
- [151] César Soto-Valero, Amine Benelallam, Nicolas Harrand, Olivier Barais, and Benoit Baudry. The emergence of software diversity in maven central. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 333–343. IEEE, 2019.
- [152] César Soto-Valero, Thomas Durieux, and Benoit Baudry. A longitudinal analysis of bloated java dependencies. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021.
- [153] César Soto-Valero, Nicolas Harrand, Martin Monperrus, and Benoit Baudry. A comprehensive study of bloated dependencies in the maven ecosystem. *Empirical Software Engineering*, 26, 2021.
- [154] Charles Spearman. *The proof and measurement of association between two things*. Appleton-Century-Crofts, 1961.

- [155] Daniel Ståhl and Jan Bosch. Experienced benefits of continuous integration in industry software product development: A case study. In *Proceedings of the 12th IASTED International Conference on Software Engineering*, 2013.
- [156] Daniel Ståhl and Jan Bosch. Industry application of continuous integration modeling: a multiple-case study. In *Proceedings of the 38th International Conference on Software Engineering Companion*, 2016.
- [157] Daniel Ståhl, Antonio Martini, and Torvald Mårtensson. Big bangs and small pops: on critical cyclomatic complexity and developer integration behavior. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 81–90. IEEE, 2019.
- [158] Margaret-Anne Storey, Brian Houck, and Thomas Zimmermann. How developers and managers define and trade productivity for quality. In *Proceedings of the 15th International Conference on Cooperative and Human Aspects of Software Engineering*, pages 26–35, 2022.
- [159] Gengyi Sun, Sarra Habchi, and Shane McIntosh. Ravenbuild: Context, relevance, and dependency aware build outcome prediction. *Proceedings of the ACM on Software Engineering*, 1(FSE):996–1018, 2024.
- [160] Xin Tan, Minghui Zhou, and Zeyu Sun. A first look at good first issues on github. In *Proceedings of the 28th Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020.
- [161] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, Akinori Ihara, and Kenichi Matsumoto. The impact of mislabelling on the performance and interpretation of defect prediction models. In *Proceedings of the 37th International Conference on Software Engineering*, 2015.
- [162] Josh Terrell, Andrew Kofink, Justin Middleton, Clarissa Rainear, Emerson Murphy-Hill, Chris Parnin, and Jon Stallings. Gender differences and bias in open source: Pull request acceptance of women versus men. *PeerJ Computer Science*, 3, 2017.
- [163] Henri Theil. Economic forecasts and policy. 1961.
- [164] Mohsen Vakilian, Raluca Sauciu, J David Morgenthaler, and Vahab Mirrokni. Automated decomposition of build targets. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 123–133. IEEE, 2015.

- [165] Roli Varma, John H Falk, and Lynn D Dierking. Challenges and opportunities: Asian women in science, technology, engineering, and mathematics. *American Behavioral Scientist*, 67, 2023.
- [166] Bogdan Vasilescu, Andrea Capiluppi, and Alexander Serebrenik. Gender, representation and online participation: A quantitative study of stackoverflow. In *Proceedings of the International Conference on Social Informatics*, 2012.
- [167] Bogdan Vasilescu, Andrea Capiluppi, and Alexander Serebrenik. Gender, representation and online participation: A quantitative study. *Interacting with Computers*, 26, 2014.
- [168] Bogdan Vasilescu, Daryl Posnett, Baishakhi Ray, Mark GJ van den Brand, Alexander Serebrenik, Premkumar Devanbu, and Vladimir Filkov. Gender and tenure diversity in github teams. In *Proceedings of the 33rd annual conference on human factors in computing systems*, 2015.
- [169] Bogdan Vasilescu, Alexander Serebrenik, and Vladimir Filkov. A data set for social diversity studies of GitHub teams. In *Proceedings of the 12th Working Conference on Mining Software Repositories, Data Track*, MSR, 2015.
- [170] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. Quality and productivity outcomes relating to continuous integration in github. In *Proceedings of the 10th joint meeting on foundations of software engineering*, 2015.
- [171] Carmine Vassallo, Sebastian Proksch, Harald C Gall, and Massimiliano Di Penta. Automated reporting of anti-patterns and decay in continuous integration. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 105–115. IEEE, 2019.
- [172] Hernan C Vazquez, J Pace, Claudia Marcos, and Santiago Vidal. Retrieving and ranking relevant javascript technologies from web repositories. *arXiv preprint arXiv:2205.15086*, 2022.
- [173] Yi Wang and David Redmiles. Implicit gender biases in professional software development: An empirical study. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Society*, 2019.

- [174] Nimmi Weeraddana, Mahmoud Alfadel, and Shane McIntosh. Characterizing timeout builds in continuous integration. *IEEE Transactions on Software Engineering*, 2024.
- [175] Nimmi Rashinika Weeraddana, Mahmoud Alfadel, and Shane McIntosh. Dependency-induced waste in continuous integration: An empirical study of unused dependencies in the npm ecosystem. *Proceedings of the ACM on Software Engineering*, 1(FSE):2632–2655, 2024.
- [176] Nimmi Rashinika Weeraddana, Sarra Habchi, and Shane McIntosh. Crash report prioritization for large-scale scheduled launches. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*, 2025.
- [177] Nimmi Rashinika Weeraddana, Xiaoyan Xu, Mahmoud Alfadel, Shane McIntosh, and Meiyappan Nagappan. An empirical comparison of ethnic and gender diversity of devops and non-devops contributions to open-source projects. *Empirical Software Engineering*, 28(6):150, 2023.
- [178] David Gray Widder, Michael Hilton, Christian Kästner, and Bogdan Vasilescu. A conceptual replication of continuous integration pain points in the context of travis ci. In *Proceedings of the 27th Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019.
- [179] Anna Wiedemann and Manuel Wiesche. Are you ready for devops? required skill set for devops teams. 2018.
- [180] Timo Wolf, Adrian Schroter, Daniela Damian, and Thanh Nguyen. Predicting build failures using social network analysis on developer communication. In *31st international conference on software engineering*, 2009.
- [181] Pavlina Wurzelová, Fabio Palomba, and Alberto Bacchelli. Characterizing women (not) contributing to open-source. In *2019 IEEE/ACM 2nd International Workshop on Gender Equality in Software Engineering (GE)*, 2019.
- [182] Junting Ye, Shuchu Han, Yifan Hu, Baris Coskun, Meizhu Liu, Hong Qin, and Steven Skiena. Nationality classification using name embeddings. In *Proceedings of the on Conference on Information and Knowledge Management*, 2017.
- [183] Zhou Yiran and Liu Yilei. The challenges and mitigation strategies of using devops during software development, 2017.

- [184] Fiorella Zampetti, Carmine Vassallo, Sebastiano Panichella, Gerardo Canfora, Harald Gall, and Massimiliano Di Penta. An empirical characterization of bad practices in continuous integration. *Empirical Software Engineering*, 25:1095–1135, 2020.
- [185] Ahmed Zerouali, Tom Mens, Jesus Gonzalez-Barahona, Alexandre Decan, Eleni Constantinou, and Gregorio Robles. A formal framework for measuring technical lag in component repositories—and its application to npm. *Journal of Software: Evolution and Process*, 31, 2019.
- [186] Chen Zhang, Bihuan Chen, Junhao Hu, Xin Peng, and Wenyun Zhao. Buildsonic: Detecting and repairing performance-related configuration smells for continuous integration builds. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–13, 2022.
- [187] Yangyang Zhao, Alexander Serebrenik, Yuming Zhou, Vladimir Filkov, and Bogdan Vasilescu. The impact of continuous integration on other software development practices: a large-scale empirical study. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 60–71. IEEE, 2017.
- [188] Markus Zimmermann, Cristian-Alexandru Staicu, Cam Tenny, and Michael Pradel. Small world with high risks: A study of security threats in the npm ecosystem. In *USENIX security symposium*, volume 17, 2019.
- [189] Irit Zohar and Miriam Belmaker. Size does matter: methodological comments on sieve size and species richness in fishbone assemblages. *Journal of Archaeological Science*, 32, 2005.