The Impact of Code Ownership of DevOps Artefacts on the Outcome of DevOps CI Builds

by

Ajiromola Kola-Olawuyi

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2023

Author's Declaration

I hereby declare that I am the sole author of this 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.

Abstract

This study focuses on factors that may influence the outcomes of CI builds triggered by commits modifying and/or adding DevOps artefacts to the projects, i.e., DevOps-related CI builds. In particular, code ownership of DevOps artefacts is one such factor that could impact DevOps-related CI builds.

There are two main strategies as suggested in prior work: **(1)** all project developers need to contribute to DevOps artefacts, and **(2)** a dedicated group of developers needs to be authoring DevOps artefacts.

To analyze which strategy works best for OSS projects, we conduct an empirical analysis on a dataset of 892,193 CircleCI builds spanning 1,689 Open-Source Software projects. First, we investigate the impact of code ownership of DevOps artefacts on the outcome of a CI build on a build level. Second, we study the impact of the Skewness of DevOps contributions on the success rate of CI builds at the project level.

Our findings reveal that, in general, larger code ownership and higher Skewness values of DevOps contributions are related to more successful build outcomes and higher rates of successful build outcomes, respectively. However, we also find that projects with low skewness values could have high build success rates if the number of developers in the project is relatively small. Thus, our results suggest that while larger software organizations are better off having dedicated DevOps developers, smaller organizations would benefit from having all developers involved in DevOps.

Acknowledgements

I express my heartfelt gratitude to all those who contributed to the realization of this thesis, foremost among them being my supervisor, Meiyappan Nagappan, and my collaborator, Nimmi Weeraddana. I sincerely appreciate your invaluable ideas, unwavering patience, and persistent support.

Dedication

This is dedicated to my friends and family.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

DevOps is a development methodology that bridges the gap between Development (Dev) and Operations (Ops) [40]. It plays a central role in projects, orchestrating effective communication and collaboration throughout the software development process. DevOps, coupled with automated deployment, is made possible through a well-defined set of practices [40]. Among various DevOps practices, *Continuous Integration (CI)* is popular because it is directly associated with the automation of the release cycles of software products. These processes hold immense importance in software development, enabling teams to streamline workflows and deliver high-quality software. In fact, large software organizations like Google and Mozilla invest millions of dollars just for CI services (excluding the cost of developers who maintain CI) [36].

However, implementing CI in software repositories can substantially burden project maintainers. The time to receive feedback from the build process could increase, and the operational costs associated with build execution also increase as a result[42].

Therefore, it is imperative to understand better the factors affecting CI systems to optimize release pipelines while lowering operational costs. To that end, many previous studies [57, 89, 46] have been aimed at understanding the factors that affect the outcome of CI builds. For example, change-set size [57], the files changed [89], the day of the week and the time of day [46] are some factors influencing the CI build outcomes. Moreover, a comprehensive understanding of these factors and those affecting the outcomes of CI builds associated with DevOps artifacts (that define DevOps pipelines) provides a competitive advantage to project maintainers. This advantage allows them to expedite the delivery of reliable solutions to end users while achieving cost efficiencies in the current fast-paced technological environment.

Several prior studies have looked into the aspects of developers who are working on DevOps artefacts of projects. For example, Wiedemann et al. [87] highlight the importance of having a dedicated team responsible for DevOps within an organization. They mention that the expertise required to manage and automate the operations of the IT infrastructure is vastly different from that needed for a software developer, and as such, individuals who possess these skills should be specifically sought after. On the other hand, major tech companies, such as Amazon, push a contradictory ideology – *you build it, you run it* [78]. This ideology advocates that all software developers responsible for building a product should also be responsible for everything required to run or operate it (including DevOps). They mention that developers who practice this ideology would also use the same creativity in supporting the applications they built, leading to a better overall experience for the end users.

The above two schools of thought are contradictory. Thus, we perform an empirical study to see if it is effective for everyone in a project to be involved in DevOps or if only a subset of people is authoring the DevOps artefacts in a project. In particular, this thesis investigates the impact of code ownership of DevOps artefacts, on the outcomes of *DevOps-related CI builds* (which we refer to as DevOps CI builds in the rest of this study). Focusing exclusively on DevOps CI builds allows for a targeted examination of the specific challenges and opportunities associated with the DevOps pipeline. DevOps CI builds are uniquely characterized by the presence of specialized artifacts that are pivotal for seamless integration, deployment, and scalability.

Therefore, by concentrating our analysis on DevOps CI builds, we aim to provide a nuanced understanding of how code ownership influences the success or failure of builds within the context of DevOps practices. This focused approach enables us to uncover insights that are directly applicable to the challenges and intricacies inherent in modern DevOps workflows, contributing to a more targeted and meaningful exploration of the research question at hand.

We conduct our analysis on a large dataset of 892,193 CircleCI[1] builds triggered from the commits related to DevOps artefacts, spanning 1,689 Open-Source Software (OSS) projects. Below, we discuss the research questions that we answer in this thesis and preview the corresponding results:

(RQ1) Does the code ownership of DevOps artefacts affect outcomes of DevOps CI builds in OSS?

Motivation and approach.

[1] https://circleci.com/

To investigate the impact of code ownership on outcomes of DevOps CI builds, we extend the definition of code ownership by Bird et al. [8] to measure ownership over time, and we term this new metric "total chronological ownership", i.e., the code ownership of an author at the time a build is triggered. We construct a mixed-effects logistic regression model [9] on a build level and investigate the relationship between total chronological ownership and DevOps CI build outcomes. Note that to avoid confounding effects, we further control the model for other factors discussed in the previous work on [89, 57], such as the number of commits associated with the build.

Results: Our mixed-effects logistic regression model shows that the code ownership of DevOps artefacts is related to the outcome of DevOps CI builds. In fact, there is a statistically significant positive relationship (coefficient = 0.173893) between the total chronological code ownership of DevOps artefacts and successful DevOps CI builds, i.e., as a developer's DevOps code ownership increases, so does the likelihood of their CI builds succeeding.

(RQ2) Does the *Skewness* of DevOps contributions affect the success rate of DevOps CI builds in OSS projects?

Motivation and Approach: Understanding the impact of code ownership on DevOps CI builds on a project level is crucial for improving the overall software development and the developer community in OSS. A more skewed distribution of DevOps contributions reflects that a majority of developers are making only minor contributions to DevOps (small code ownership) while a few developers are making significant contributions to DevOps (large code ownership), and vice versa. We use a linear regression model to study the relationship between the *Skewness* of DevOps contributions and the success rate of DevOps CI builds. We further control this model for features, such as the number of DevOps authors and the number of DevOps commits.

Results: Our results show that the *Skewness* of DevOps contributions in OSS projects has a statistically significant positive impact on projects' success rate of DevOps builds. We find that the linear regression coefficient is 1.28 for modelling the rate of successful DevOps CI builds. Thus, the higher the *Skewness*, the higher the successful rate of DevOps CI builds in projects. Accordingly, confining DevOps-related changes to a specific group of developers enhances the likelihood of successful builds. Therefore, project maintainers may actively promote the idea that fewer people should be responsible for maintaining most of the DevOps artefacts.

Our study reveals that code ownership of DevOps artefacts strongly impacts the outcome of DevOps CI builds and informs project maintainers to take cognizance of our findings while forming development teams.

The three major contributions of our study are (1) an in-depth analysis of build-level code ownership of DevOps artefacts, (2) an in-depth analysis of project-level code ownership of DevOps artefacts, (3) a large dataset of 892,193 DevOps CI builds and the build-level and project-level features and the scripts to reproduce this dataset and our statistical analyses (RQ1 and RQ2).[2]

1.1 Key Definitions

In this section, we present the terminologies and corresponding definitions we will use throughout this thesis.

DevOps Artefact. A file pertaining to a DevOps tool,[3] e.g., the `Dockerfile` is a DevOps artefact pertaining to a containerization tool called Docker.[4]

DevOps Commit. A git commit that modifies/adds or deletes at least one DevOps artefact, e.g., a commit (hash `#da500aa`) that contains changes to four files, including a `Dockerfile`.

DevOps Build. A CI build associated with at least one DevOps commit, e.g., a CI build that contains a few commits, including the above commit (`#da500aa` that modify a `Dockerfile`).

[2] https://zenodo.org/records/10146006
[3] https://periodictable.digital.ai/
[4] https://www.docker.com/

Chapter 2

Experimental Design

This chapter describes our process for collecting and curating the dataset we use to address our research questions. In Figure 2.1, we provide an overview of our study design, which we detail below.

2.1 Data Preparation

Our study integrates data from various sources (i.e., GitHub API and Gallaba et al. [24]) to acquire the data needed for our analysis. Below, we describe the data preparation steps in detail.

2.1.1 Filtering the dataset

We begin with the dataset of CircleCI build records collected by Gallaba et al. [24]. This dataset contains 22 million CircleCI builds across 7,795 GitHub repositories for eight years. We first perform project-level filtering to eliminate *toy projects*, inspired by previous work [54, 43], by using thresholds of the number of commits and the number of builds.

- **Number of commits:** Fig. 2.2 plots candidate threshold values of the number of commits against the number of surviving projects. We selected a threshold of 250 commits because it is closer to a "knee" in the curve. This threshold selects 3,247 projects.

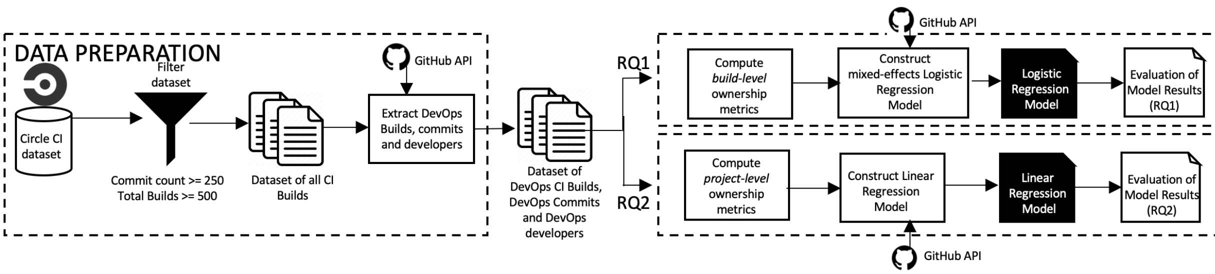

Figure 2.1: Experimental design overview.

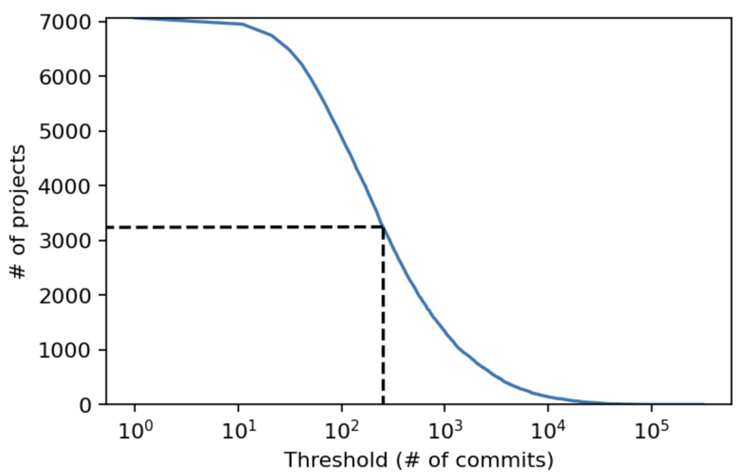

Figure 2.2: Commit thresholds.

- **Number of builds:** In Fig. 2.3, we illustrate candidate threshold values for the number of builds against the number of surviving projects, with a threshold of 500 builds chosen due to its proximity to a "knee" in the curve, resulting in a dataset of 17,304,451 builds across 1,794 projects.

The dataset provided by Gallaba et al. [24] contains the outcome for every build, which could be one of the following: *success*, *failed*, *timeout*, *infrastructure fail*, etc. We select the builds with either the *success* status or *failed* status.

2.1.2 Retrieving commit data

After filtering the projects and builds of interest, the next step is extracting commit data associated with builds. The original dataset provided by Gallaba et al. [24] contains

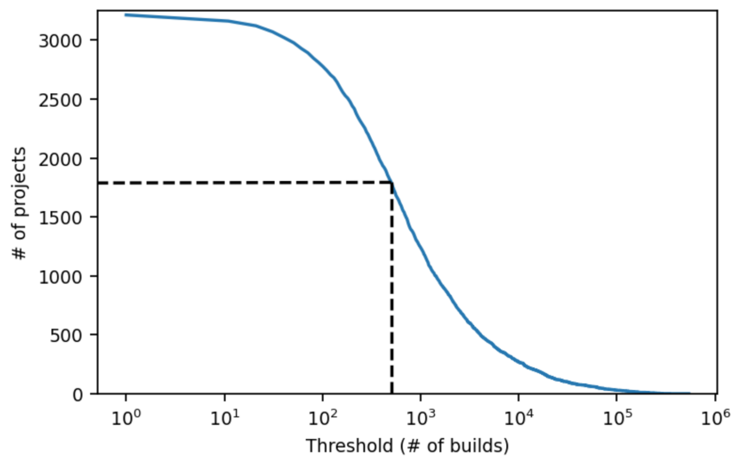

Figure 2.3: Build thresholds.

DevOps Category	Regular Expression	DevOps Tool
Continuous Integration	.*[.]circleci/config[.]yml$	CircleCI
	.*[.]github/workflows/.*	GitHub Actions
Containers	.*[.]dockerfile$	Docker
	.*mesos-master[.]sh$	MesOs
Deployment	*[.]gocd[.]yml$	GoCD
	.*appspec[.]yml	AWS Code Deploy
Configuration	.*[.]tf$	Terraform
	.*ansible[.]cfg$	Ansible

Table 2.1: Some regex patterns for detecting DevOps files.

commit information associated with each build in the dataset. In addition, we need the files associated with the change sets of commits to identify whether a certain commit is a DevOps commit. Gallaba et al.'s dataset does not contain the files changed in commits. Thus, we use the GitHub API to extract the files in the change set of each commit.

2.1.3 Identifying DevOps-related commits and builds

To filter DevOps commits and builds, we analyze which files are DevOps files in the change sets of commits associated with builds. If there is a DevOps file in a commit, we consider it as a *DevOps commit*, and if there is any DevOps commit associated with a build, we

Build	Build Number	Outcome
DevOps Build 1	t	Failed
DevOps Build 2	t+1	Success

Table 2.2: Failed DevOps CI builds.

consider that build as a *DevOps CI build*.

To identify DevOps files, we rely on the periodic table of DevOps tools published by Xebia labs[1] for a comprehensive list of all DevOps tools and the categories of their usage. Since there is a large variety of DevOps tools, we select the tools used for Continuous Integration, Deployment, Containers, and Configuration because such practices are commonly used in projects that use DevOps [74]. These DevOps tools include Kubernetes, Docker, Travis CI, etc.

Next, we review the above tools' documentation and list the filenames or file extensions of their configuration files. Then, we build a DevOps file classifier based on a regular-expression (Regex) search to detect DevOps artefacts by examining the filenames, file extensions, and directories in which they are stored. Table 2.1 shows regex patterns we use and the corresponding tools they detect. To ensure the validity of this classifier, we manually checked a sample of files. We found that the Cohen's Kappa agreement level [11] is 0.82 between the coder and the classifier, showing a strong agreement. Then, we use this DevOps file classifier to analyze all the files in every commit in every build in our filtered project dataset.

We find 444,630 DevOps commits and 1,467,710 DevOps builds. Five projects did not have any DevOps builds, so we remove them from the study, reducing the number of projects in our dataset to 1,789. To be certain that build failures in our dataset resulted from changes to DevOps artefacts within the build and not source files, we consider a failed build as a failed "DevOps" build if and only if there is a successful DevOps build observing right after it. We assume that the failed DevOps builds, in this case, are indicative of issues with DevOps artefacts and a subsequent successful build signifies a deliberate response by the development team to address and rectify the identified issues within the DevOps process. Table 2.2 describes what we define as a failed DevOps build.

We drop all other failed builds with DevOps commits that do not meet this criterion. To mitigate the bias of the previous build's outcome introduced by this step, we excluded

[1]https://periodictable.digital.ai/

the successful builds immediately after failed DevOps builds. Doing so reduced the number of builds in our dataset to 996,924 across 1,789 projects.

2.1.4 Merging author identities

GitHub allows multiple names and email addresses to be associated with the same GitHub account. As a result, we notice that some commits within a project have very similar names and are most likely the same person. For example, some commits could be contributed by *Mark Smith* while others would be contributed by *M. Smith*. To account for this, we write a script to merge similar author names. We leverage previous work on string matching author names [21, 14] and computed the `jaro-winkler` similarity between the author names. If this value was greater than 0.9 [14], we assume the authors are the same and merge their contributions. To ascertain that this script works as it should, we manually merge the authors from 92 (minimum sample size) random repositories and found a Cohen's Kappa agreement score [11] of 0.91 with our script, which shows a strong agreement.

2.1.5 Identifying DevOps developers

We extract the DevOps developers who authored the DevOps commits in our dataset. More specifically, we define a DevOps developer as one that makes at least one DevOps commit to the project. We group all DevOps commits in each project by the author names to see all the authors and the corresponding number of DevOps commits they made.

We exclude repositories with only one developer from our dataset. This results from the bias they introduce as the chronological ownership of every build in such repositories would be 100%, and it is impossible to compute the Skewness of DevOps contributions by authors if there is just one author. This further reduces the number of repositories in our dataset from 1,789 to 1,689, and DevOps builds to 978,009. We also drop builds that have more than one DevOps contributor. This reduces the number of DevOps builds to 892,193. We decide to drop these builds as they are a small subset of our dataset i.e. about ten percent.

We also exclude commits made by bots from this analysis. To do this, we built a regex-based *bot classifier* that checked if the author of a commit was a bot or not. We evaluated the performance of this bot classifier by manually classifying 400 author names and computing the Cohen's Kappa agreement with it. We obtained a score of 0.88, which shows a strong agreement.

2.2 RQ1 Design of the Build-Level Analysis

In our RQ1, we analyze the build-level impact of the DevOps code ownership on build outcomes. Below, we detail the steps.

2.2.1 Computing build-level DevOps ownership

Following Bird et al.'s [8] definition of code ownership, we define chronological code ownership of DevOps artefacts as the percentage of the number of commits made by a DevOps developer to the total number of DevOps commits in the project up until when the build is triggered. For instance, the first DevOps build that an author *(author A)* in a project triggers would have a chronological ownership value of 100%, provided all DevOps commits before the time the build was triggered were made by the same author. Similarly, assuming another author *(author B)* makes a DevOps commit that triggers another DevOps build. If there are ten DevOps commits in total at the time of this build, *author B*'s chronological ownership would be 10%.

2.2.2 Constructing the logistic regression model

Based on previous work [42, 90, 57], we collect 17 features to serve as control features to our model. Some of these features include the number of commits in the build, the number of lines of code changed, the time the build was triggered, etc. We first eliminate colinearity from the features by leveraging the `varclus` function in the `Hmisc` [2] package (in *R*). A total of 16 control features survived the analysis. We build an initial model with these features, and then we analyze the Variance Inflation Factors (VIF) to detect multicollinearity [6]. Following the previous work by Nadri et al. [55], we remove features with VIF values greater than three. Table 2.3 shows the 14 features that survive. Using these features, we build a mixed-effects logistic regression model to investigate the relationship between the `chronological ownership` of DevOps artefacts and the outcome of DevOps CI builds. Moreover, the use of this "mixed-effects" model accounts for the hierarchical nature of the data by including the name of the repositories as a random effect.

[2] https://www.rdocumentation.org/packages/Hmisc/versions/5.1-1

Category	**Feature**	**Definition**
Change size (inspired by [46])	`number_of_devops_commits`	Number of DevOps commits in the build
	`devops_change_size`	Number of lines changes in DevOps artefacts
Files Changed [46, 90, 89]	`gh_diff_files_removed`	Number of files removed from the build
	`num_of_devops_artefacts`	Number of DevOps artefacts in the build
Link to Last Build [33, 57]	`prev_built_result`	Outcome of the last build
	`same_committer`	Boolean value showing if the last build was triggered by the same developer
Triggering Commit [46, 33, 73]	`gh_is_pr`	Boolean value showing if the build is part of a Pull Request
	`day_week`	Day of the week the build was triggered
	`time_of_day`	Time of the day the build was triggered
Cooperation [90]	`num_of_distinct_authors`	Number of distinct authors in the build
Committer History [57, 65]	`committer_fail_history`	The fail rate of the builds by the current committer in the past
	`committer_recent_fail_history`	The fail rate of the last five builds by the current committer
	`committers_avg_exp`	Average number of builds per committer at the time the build was triggered
Ownership	`total_chronological_ownership`	Percentage of DevOps commits made by current committer to the total number of DevOps commits at the time of the build

Table 2.3: Features used in the mixed-effects logistic regression model.

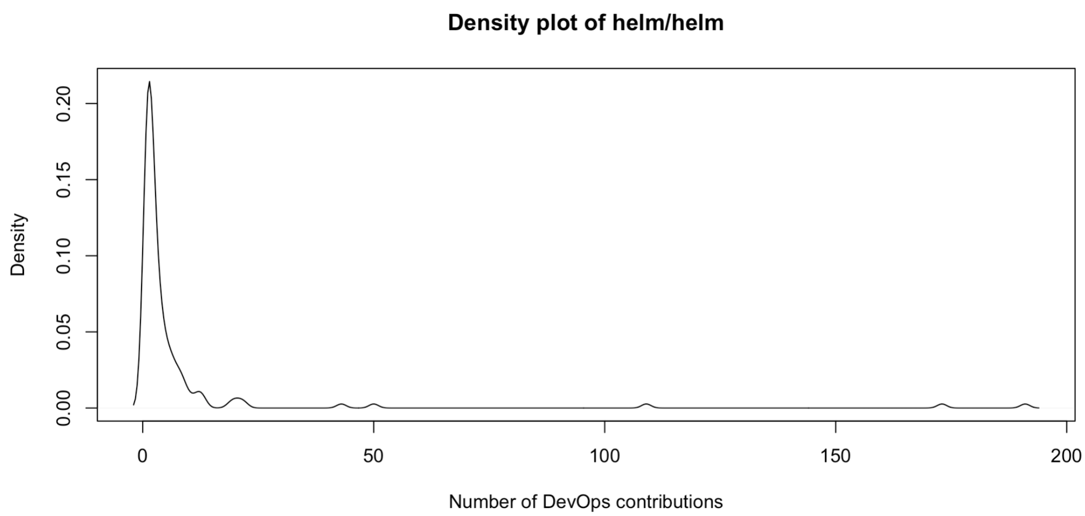

Figure 2.4: Density plot of DevOps contributions (right-skewed) in helm/helm.

2.3 RQ2 Design of the Project-Level Analysis

Understanding code ownership is not only about who owns what but also unveils insights into how developers contribute to DevOps tasks across the entire project. Our RQ2 delves into the lasting impact of code ownership during the build stage, exploring whether it creates a cascading effect that shapes the distribution of DevOps contributions among authors at the project level. This investigation aims to uncover how optimizing code ownership can seamlessly enhance collaborative efforts and overall project efficiency. Below, we detail our analysis.

2.3.1 *Computing project-level DevOps ownership*

We compute the *Skewness*[3] of the distribution of DevOps contributions made by DevOps developers for every project in our dataset. The `Skewness` measures how distorted a particular distribution is from the normal distribution. A positive *Skewness* value corresponds to a right-skewed distribution, and a negative *Skewness* value corresponds to a left-skewed distribution. A *Skewness* value of zero reflects a normal distribution.

[3] https://www.scribbr.com/statistics/skewness/

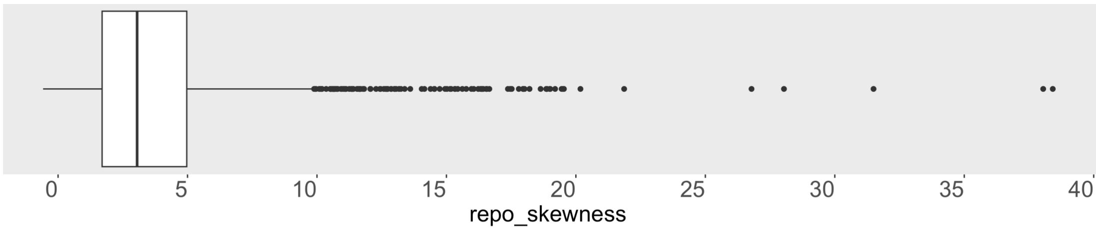

Figure 2.5: Boxplot of the Skewness to DevOps contributions.

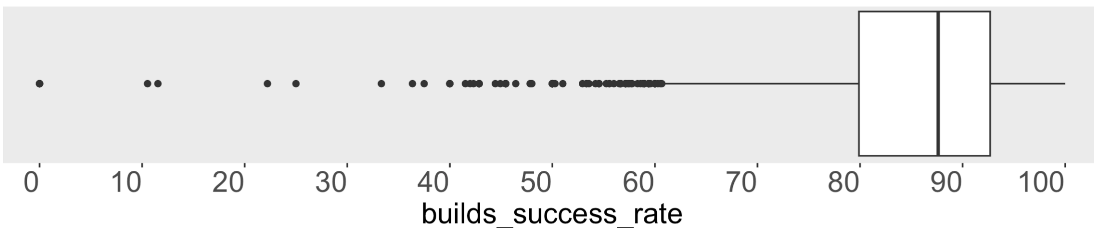

Figure 2.6: Boxplot of the rate of successful DevOps builds.

Fig. 2.4 shows the helm/helm project's density plot[4] corresponding to the number of contributions made by authors on DevOps commits. The x-axis shows the number of DevOps contributions, and the y-axis shows the *density* of developers. This figure shows an overwhelming majority of developers who made a few DevOps contributions. Thus, the distribution of the author contributions is right-skewed, and the *Skewness* value is 6.49 (+ve). Such a right-skewed distribution indicates that a majority of the developers in the project made a relatively small number of contributions to DevOps, while a few developers made most of the contributions. This is particularly relevant to our definition of code ownership of DevOps artefacts because it shows that a few developers in the project have significantly higher ownership of DevOps artefacts than the rest. This also implies that a dedicated group of DevOps developers may maintain DevOps artefacts in the project. Similarly, a left-skewed distribution indicates that the project does not have a potential group of developers working on DevOps artefacts, (i.e., all developers, in general, make both DevOps and non-DevOps contributions. We use this *Skewness* as a proxy to project-level DevOps code ownership, and investigate its impact on the failure rate of DevOps CI builds in the project.

[4]A density plot is a representation of the distribution of a numeric variable. It uses a kernel density estimate to show the probability density function of the variable

Table 2.4: Features used in the linear regression model.

Feature	Definition
`number_of_repo_builds`[75]	Total number of builds project in the project
`number_of_devops_artefacts`	Total number of DevOps artefacts in the project
`project_maturity` [62, 55]	Number of days since the first build was made
`recent_failure_rate`	Failure rate of last five builds in the project
`commits_per_build`	Average number of commits per build in the project
`number_of_devops_authors` [4]	Total number of DevOps developers in the project
`forks` [62, 55, 4]	Total number of forks on GitHub [5]
`repo_skewness`	Skewness of developer contributions to DevOps in the project

2.3.2 *Constructing the linear regression model*

We use a linear regression model to model the impact of project-level code ownership (i.e., Skewness) on project-level CI build success rate. To obtain the success rate of each project in our dataset, we express the number of successful DevOps builds over the total number of DevOps builds as a percentage. Since the Skewness is not the only feature influencing DevOps build failure rate, we also collect several project-level control features, such as the number of stars, forks, contributors, etc., based on the prior studies [93, 37].

We first remove collinearity and multicollinearity among the extracted features. Table 2.4 shows the surviving features after removing collinearity and multicollinearity. Next, we pass these features into a linear regression model. We present results in Section 3.

Challenges of extracting our datasets for the analyses needed for RQ1 and RQ2. Identifying DevOps commits and extracting the features needed to construct the two models are challenging and demanding. For each of the 15,655,048 CircleCI builds in our initial dataset, we call the GitHub API for every commit within the build, which is costly. To speed up the data collection, we employ multiprocessing[6] for querying and fetching the results from the GitHub API. This enables us to parallelize the API calls, processing multiple requests simultaneously. The resource usage (in total) amounts to nearly 7.6 CPU core years. It took us about ten months to gather the data completely.

[6]https://docs.python.org/3/library/multiprocessing.html

Chapter 3

Results

3.1 (RQ1) Does the code ownership of DevOps artefacts affect the outcome of DevOps CI builds in OSS?

To analyze the impact of chronological code ownership of DevOps artefact on DevOps build outcomes, we construct a mixed-effects logistic regression model (Section 2.2). The model's performance is evaluated using conditional R^2. In particular, the conditional R^2 metric measures the total variance explained by both fixed and random effects. Our model achieved a value of 0.74, indicating that it explains approximately 74% of the variance with both fixed and random effects considered.

In Table 3.1, we present the impact of different features (including the chronological code ownership) on the outcomes of DevOps CI builds. The *coefficients*' magnitudes signify their strength, while the *coefficients*' signs indicate the direction of the relationship of the corresponding feature and the DevOps CI build outcome. A negative coefficient shows that as the feature increases, the dependent variable (the DevOps build outcome) becomes more likely to be a failure and vice versa. Conversely, a larger magnitude shows that a feature has a stronger impact on the DevOps CI build outcome and vice versa. Note that we also show the statistical significance levels of all the coefficients in the *p-value* column.

In addition, following the previous work [55, 62], we compute the odds ratio of each feature to better understand the relationship between the dependent and independent variables. We show the odds ratios in the *Odds* column of Table 3.1. The interpretation of the odds ratio depends on the type of variable. In particular, for continuous variables, the odds ratio shows the chances of the dependent variable occurring per unit change in the independent variable, while for categorical variables, the odds ratio shows the chances of the dependent variable occurring compared to the default value of the categorical variable [55]. Below, we describe the main observations from the table and the corresponding discussions.

> ***Observation 1:*** *Higher levels of chronological code ownership of DevOps artefacts in a build leads to more successful DevOps CI builds.*

Chronological code ownership (build-level code ownership) of a DevOps CI build is the percentage of the number of commits made by a DevOps developer to the total number of DevOps commits in the project up until when the build is triggered (Section 2.2.1). Table 3.1 shows that the chronological ownership of DevOps artefacts has a statistically significant impact ($p<0.001$) on the outcome of DevOps CI builds. Its positive coefficient indicates that a high code ownership is associated with an increased probability of achieving a successful build outcome. From the odds ratio of the total chronological ownership (`exp(0.173893) = 1.19`), we can deduce that the chances of a successful build would increase by 19% for every percentage increase in an author's DevOps commits relative to the total number of DevOps commits in the project.

Furthermore, we use the ANOVA test [69] to determine how much of the variance in build outcomes can be explained by total chronological ownership. We found that code ownership of DevOps artefacts explains 0.69% of the variance of DevOps CI builds' build outcomes. While this percentage of explained variance may be modest, the model coefficients establish a robust and statistically significant correlation between higher levels of code ownership in DevOps artefacts and increased success in DevOps CI builds, underscoring the impact of code ownership on the overall outcomes of DevOps CI builds. This observation about code ownership complements Wiedemann et al.'s [87] work, which emphasizes the positive correlation between a concentrated group of developers overseeing DevOps artifacts and enhanced success in DevOps CI builds.

Despite observing a positive relationship between code ownership and DevOps build outcomes, a close inspection of our dataset reveals several instances where DevOps builds deviate from this trend. To delve deeper into such deviations, we further analyze builds that failed with high code ownership and those that are successful with low code ownership. We consider builds with high code ownership to be those with code ownership within the upper quartile (top 25%) and builds with low ownership to be those with ownership within the lower quartile (bottom 25%).

Discussion 1: *DevOps builds that fail despite having high levels of chronological code ownership are early builds and vice versa.*

A project's first set of DevOps CI builds typically has higher ownership because fewer authors have made contributions at the time. For example, suppose there are ten DevOps commits in a project; six of these commits are made by *committer A*, while the remaining four are made by *committer B*. If a build is triggered by *committer A*, then this build would have chronological ownership of 60%; in the opposite case, where that build is submitted by *committer B*, the chronological ownership value would be 40%, which are both substantial levels of chronological ownership.

Our analysis shows 24,971 failing builds with such high ownership levels. These builds are concentrated in the early stages of repository timelines, often associated with the initial configuration of CI pipelines, making them more prone to errors.

For builds failing with high code ownership, we find that the mean and median values for the `number_of_builds_before_a_build` are 511.6 and 113, respectively. In contrast, when considering the entire dataset, these values are notably higher at 5,847 and 1,467, underscoring the unique characteristics of builds with high code ownership. An illustrative case is the `#60032dd1e1d87f460fac092f` build of the pankona/gomo-simra project, with chronological code ownership of 93.33% at the time but fails; the number of builds before that particular build is 37, which is much smaller than the mean and the median of the entire dataset.

Likewise, the rationale extends to successful builds despite low ownership levels. We find 204,951 such builds in our dataset, which often appear *later* in the repository timelines, depicting cases where the pipeline must have been set thoroughly and extensively tested. We find that, for builds in this category, the `number_of_builds _before_a_build` is much higher than that of the entire dataset; the mean and median are 13,853 and 6,467, respectively, clearly surpassing the corresponding values for the whole dataset, which are 5,847 and 1,467.

Observation 2: *As the number of DevOps artefacts in a build increases, the likelihood of a successful build decreases.*

Another interesting observation we made from our model (Table 3.1) is that the number of DevOps artefacts in a build also has a statistically significant effect ($p<0.01$) on the outcome of the build. Its coefficient indicates a negative relationship between the number of DevOps artefacts and the dependent variable (the build outcome), i.e., as the number of

DevOps artefacts increases, the build's success becomes less likely. In addition, the odds ratio of this feature (`exp(-0.039625)=0.96`) shows that for every extra DevOps artefact in the build, there is a 4% decrease in the chances that the build succeeds. A similar trend is also observed in the DevOps change size feature, which measures the number of lines changed in all DevOps artefacts in the build. Table 3.1 shows that there is a 5% (`exp(-0.050963)=0.95`) decrease in the chances of success of a build for every line changed in a DevOps artefact. This observation is in line with the prior work of Luo et al. [46], who modeled CI build outcomes (regardless of being related to DevOps files). They found that as more files are changed in a build, there is less chance for that build to result in a successful one.

Observation 3: *As the committers' average experience increases, the likelihood of the DevOps CI build's outcome being a success also increases.*

Table 3.1 shows that the average number of builds of committers at the time of the current build (committers' average experience) has a statistically significant ($p<0.001$) effect on the outcome of DevOps CI builds. Its positive coefficient shows that provided all other features remain constant, as the committers' average experience increases, the chances of that build's success also increase.

Our results further show that the committers' average experience explains about 44% of the variance of the DevOps CI build outcome by conducting the ANOVA test. We believe this is because the committers' average experience is pivotal in capturing the typical instability observed during the initial stages of build creation, where inexperienced contributors may encounter challenges. It is not uncommon for early builds to exhibit failures as they are being set up. It is also worth noting that the committers' average experience is a *historical statistic* [57] and Ni et al. [57] reports that historical statistical features are the most critical features in predicting the outcome of a build.

Summary RQ1: *On a build level, code ownership of DevOps artifacts does matter. The result of our experiments distinctly reveal a positive relationship between the code ownership of DevOps artifacts and outcomes of DevOps CI builds.*

3.2 (RQ2) Does the *Skewness* of DevOps contributions affect the success rate of DevOps CI builds in OSS?

We construct a linear regression model (Section 2.3) to investigate the project-level impact of code ownership (i.e, the Skewness of DevOps contributions) on the DevOps build success rate of the 1,689 projects in our dataset. Table 3.2 shows the model results. As similar to Table 3.1 in RQ1, the magnitude and sign of the *Coefficient* column in Table 3.2 represents the strength and direction of the relationship between the feature and the dependent variable (build success rate), respectively. From the table, we make the below observations.

> ***Observation 4:*** *A high Skewness of DevOps contributions within a project is related to elevated success rates in DevOps builds in that project.*

From Table 3.2, we observe that the Skewness of DevOps contributions has a statistically significant (*p-value*$<$*0.05*) effect on the success rate of DevOps CI builds. We also use the ANOVA test [69] to determine how much of the variance in the success rate of DevOps builds is explained by the Skewness of DevOps contributions, and we find this value to be about 2.8%. The model coefficient of the Skewness is positive, indicating that as the Skewness of DevOps contributions in a project increases, the rate of successful DevOps builds increases. For example, the helm/helm project shown in Figure 2.4 (Section 2) has a Skewness of 6.49; the success rate of DevOps CI builds in this project is 91.38%. Furthermore, from figures 2.5 and 2.6 (in Section 2), we see that the median Skewness and success rate of DevOps builds are 3.05 and 87.62 respectively. Thus, helm/helm has a high Skewness and a high success rate of DevOps CI builds.

Although highly skewed DevOps contributions elevate the success rate of DevOps builds in projects, we still find some projects in our dataset that deviate from this trend. We perform an extended analysis, investigating the characteristics of the deviated projects to understand the underlying reasons for such deviations better. In particular, we investigate projects with high Skewness values and low build success rates as well as projects with low Skewness values and high success rates. We consider projects with high Skewness values to be the ones with Skewness values in the upper quartile, i.e., the top 25%; on the other hand, we consider the projects with low Skewness values to be the ones with Skewness values in the lower quartile (bottom 25%). Similarly, we obtain projects with high Skewness and low build success rates. This analysis reveals that 78 projects have high Skewness values and low build success rates.

Discussion 2: *Projects with high Skewness of DevOps contributions and low (DevOps CI) build success rates have more complex builds.*

By delving deeper into the attributes of these projects, we find that they have more intricate builds, i.e., builds with a high average number of lines changed per build. The mean and median of the average number of lines changed per build in a project with high Skewness values with low build success rates are 36,015 and 1,944, respectively, and are much higher than those for the projects in our entire dataset (mean and median of the whole dataset are 12,167 and median 1,247, respectively).

On the other hand, we find 97 projects with low Skewness values and high build success rates. These projects have a much lower number of DevOps developers.

Discussion 3: *Projects with low Skewness values and high build success rates have a smaller number of developers.*

For example, hanami/validations project has a Skewness of 0.707, a build success rate of 95.41%, and has just three DevOps developers. In general, among the projects with low Skewness values and high build success rates, we find that the mean and median number of developers are four and six, respectively, which are much less than the mean (61) and median (20) values of the entire dataset. The fact that these projects have high build success rates despite having a smaller number of developers implies that when a project has a small number of developers, a more balanced allocation of contributions may improve the DevOps build success rate.

Because the number of developers in a project impacts the DevOps build success rate, we decide to look at the distribution of the number of developers in the projects that fully satisfy our Observation 4, i.e., the projects with high Skewness values and high build success rates.

Discussion 4: *Although the total number of developers that contribute to projects with high Skewness values and high build success rates is large, the number of DevOps developers that make most of the DevOps commits is relatively small.*

This analysis reveals that the number of DevOps developers in such projects is much larger compared to our entire dataset and to the projects with low Skewness values and high success rates. In particular, the mean and median of the number of developers for projects with high Skewness values and high success rates are 222 and 113, respectively.

Furthermore, in the projects with high Skewness values and high build success rates, to examine how many developers are accountable for the majority of the DevOps commits, we retrieve the number of developers in each project who are responsible for 80% of the total DevOps commits in the project. We find that the mean and median numbers of DevOps developers contributing to 80% of the DevOps commits made in these projects are seven and three, respectively. This reinforces the idea that, despite a sizable developer pool, a select few shoulder most changes in DevOps artefacts (Observation 1 in RQ1), resulting in increased ownership and success in DevOps CI builds.

> ***Summary RQ2:*** *On a project level, the Skewness of DevOps contributions does affect the success rate of passed DevOps CI builds. Our results show a statistically significant positive relationship between the two.*

Table 3.1: Results of the build-level mixed-effects logistic regression model.

Feature	**Coefficient**	***P-Value***	**Odds**
number_of_devops_ commits	0.034758	3.32e-10 ***	1.04
devops_change_size	-0.050963	1.22e-12 ***	0.95
prev_built_resultcanceled	-0.383202	3.54e-09 ***	0.68
prev_built_resultfailed	-2.031670	< 2e-16 ***	0.13
prev_built_result_infrastructure_fail	0.338537	0.096438 .	1.40
prev_built_resultno_tests	0.281516	0.054136 .	1.33
prev_built_resultsuccess	0.485515	< 2e-16 ***	1.63
prev_built_resulttimedout	-0.035349	0.834786	0.97
num_of_distinct_authors	0.002759	0.612027	1.00
num_of_devops_artefacts	-0.039625	1.94e-08 ***	0.96
gh_diff_files_removed	0.019927	0.000883 ***	1.02
gh_is_prTrue	-0.413991	< 2e-16 ***	0.66
day_weekMonday	-0.050158	0.003287 **	0.95
day_weekSaturday	-0.096326	2.75e-05 ***	0.91
day_weekSunday	-0.022346	0.344655	0.98
day_weekThursday	-0.026590	0.110966	0.97
day_weekTuesday	-0.014659	0.377967	0.98
day_weekWednesday	-0.060893	0.000232 ***	0.94
time_of_day	0.001523	0.758338	1.00
same_committerTrue	-0.461960	< 2e-16 ***	0.63
committer_fail_history	-0.035224	3.92e-06 ***	0.96
committer_recent_fail_history	-1.220983	< 2e-16 ***	0.29
committer_avg_exp	1.837764	< 2e-16 ***	6.28
total_chronological_ownership	**0.173893**	**< 2e-16 *** **	**1.19**

Significance codes: 0: ***, 0.001: **, 0.05: *

Table 3.2: Results of the project-level linear regression model.

Feature	**Coefficient**	***P-Value***
number_of_repo_builds	5.148e-05	0.000244 ***
number_of_devops_files	-2.745e-04	0.931195
project_maturity	-7.436e-04	0.177558
recent_failure_rate	-1.344e-01	<2e-16 ***
commits_per_build	-2.092e-02	0.521927
number_of_devops_authors	-5.267e-04	0.904448
forks	-6.500e-05	0.711234
repo_skewness	**2.286e-01**	**0.019384 ***

Significance codes: 0: ***, 0.001: **, 0.05: *

Chapter 4

Threats To Validity

4.1 Construct Validity

We use a regular expression-based classifier to decide whether a code artefact is a DevOps artefact. This also cascades into our definitions for DevOps commits and DevOps builds. We rely solely on the project's file names, extensions, and directories to make our classification. Relying on a regular expression-based classifier could lead to misclassifying artefacts, when distinguishing between DevOps and other file types. To mitigate this, we randomly select a sample of 400 files within our dataset and manually classify these files. We then compute the Cohen's Kappa agreement score [11] between the file classifier and the coder. We find an agreement level of 0.82, which indicates a near-perfect agreement.

In addition, as a result of GitHub allowing developers to use aliases, the same author in a project could make commits with different names.[1,2] This poses a challenge for quantifying each author's DevOps contributions to the project. To mitigate this threat, we leverage previous work [21, 14] and compute the `jaro-winkler` similarity between author names. If the calculated similarity exceeds 0.9 [14], we assume they are the same author and sum up the contributions. To ensure the validity of this method to merge aliases, we examined all authors from a random sample of projects within our dataset and manually merged authors. We also computed the Cohen's Kappa score between the `jaro-winkler` method and the coder and got a value of 0.91, indicating a near-perfect agreement.

[1] `https://git-scm.com/book/en/v2/Git-Basics-Git-Aliases`

[2] 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

4.2 Internal Validity

We may have missed confounding factors that could impact the interpretations of our results, e.g., we observe that the previous build outcome has a significant impact on the current build outcome (Table 3.1 in Section 3.2). However, the true root cause for this, in the context of failing builds, may be that developers retry failed builds, assuming there is some flakiness in the build [7]. We call them *failure streaks*, i.e., a series of arbitrarily retried builds after a failure without making any changes. Whether our features share causal or correlational relationships with DevOps build outcomes is a potential direction for future work.

Our definition of DevOps builds may be biased. We define DevOps commits as commits that add and/or modify DevOps artefacts; considering the builds triggered by DevOps commits. This is because a build may be associated with DevOps commits and other commits, and thus, the build's outcome could result from a commit that is not a DevOps commit. Consequently, the failure of a build that we consider a DevOps build may not be failed because of a DevOps commit but because of another commit. To ensure that we only consider DevOps builds that failed only because of a DevOps commit (and not because of other commits), we consider failed DevOps builds as the DevOps builds preceded by a successful DevOps build, as depicted in Table 2.2 in Section 2.1.3. Doing so also mitigates biases in our results, for example, due to *failure streaks*.

4.3 External Validity

Our models rely on data from projects executed on CircleCI, so our results may not be reflected on other CI services. Nonetheless, given their shared characteristics, we believe that the outcomes of our study could be adapted to other CI services. For instance, the utilization of YAML and the general structure of pipelines are often consistent across these services. However, we also recognize the need for future studies focusing on other CI services to ensure comprehensive insights.

Chapter 5

Related Work

5.1 DevOps Technologies

In contemporary software engineering, the emergence of DevOps represents a relatively recent but pivotal concept. Several studies [39, 12, 74, 18] defined the term DevOps. For example, Jabbari et al. [39] defined DevOps as "a development methodology aimed at bridging the gap between Development and Operations." Among several benefits of adopting DevOps, the major one is that it forces the development and operations teams to interact with each other more than before, leading to enhanced collaboration and communication [64].

However, adopting DevOps is not without its limitations; several studies [30, 13, 74, 71, 85], thus, focused on the problems in adopting DevOps practices in organizations. Grande et al. [30] conducted a systematic literature review to investigate the challenges of practicing DevOps in globally distributed teams. They found the following challenges: the complex and extensive skillset required, communication challenges between the developers and operations team, and employee resistance to change. Diel et al. [13] focused solely on the communication challenges and reported that the geographical distance between team members and their frequency of interaction could influence the communication gap. In addition, other studies [79, 34] discussed the challenges of configuring complex DevOps files. In particular, Tamraw et al. [79] and Henkel et al. [34] discussed the challenges of configuring build files (e.g., `Makefile`s) and writing `Dockerfile`s, respectively. These studies further suggested tools to help developers work on such files.

Other work studied the tasks of DevOps developers [44, 88]. For example, Kerzazi et al. [44] conducted an empirical analysis of online job postings, aiming to identify and compare the primary responsibilities of DevOps engineers. This analysis revealed that *automation* is the essential activity expected by DevOps developers, which is further explored in several other studies [53, 86].

While many of the above studies explore various facets of DevOps, our work takes a distinct approach by examining DevOps from a different perspective. Specifically, our work shed light on the quantitative impact of code ownership of DevOps artefacts and the Skewness of DevOps contributions in a project which has not been extensively studied.

5.2 Continuous Integration (CI)

CI is a DevOps practice that frequently integrates code changes into a shared code base [16]. CI provides several advantages to the software teams that adopt it [82, 36]. For example, Vasilescu et al. [82] studied a dataset of 246 OSS projects that use CI. They found that CI improves the productivity of project teams. While CI is advantageous for projects, other studies revealed the difficulties of adopting CI. Among the several challenges that come with CI, build failures are a challenge that is unavoidable for many projects [60, 59, 83, 45, 52, 84, 76, 25]. For example, Kerzazi et al. [45] analyzed 3,214 builds produced in a large software company over a period of 6 months. They found that a substantial proportion (17.9%) of builds are failing. In addition, Vassallo et al. [83] analyzed build failures in 349 OSS projects and 418 proprietary projects. Furthermore, this study revealed, for both the OSS and proprietary projects, the overall percentage of failing builds during the period of observation is 26%, which is much higher compared to Kerzazi et al. [45]'s results. This analysis showed that failures related to deployments and release preparation are among the key categories of build failures.

Due to challenges of observing build failures, several studies [46, 65, 33, 90, 89, 57, 10, 66] focused on predicting build outcomes in advance. For example, Luo et al. [46] constructed four build outcome prediction models for the TravisTorrent dataset. They found that the *number of commits in a build* is the most critical factor determining the probability of build failures. Hassan et al. [33] conducted a similar study on the same dataset but restricted their investigation to projects using Ant, Gradle, and Maven. They utilized a random forest model and found the outcome of the previous build to be the most crucial factor in predicting build outcomes. Furthermore, Saidani et al. [65] used deep learning to predict CI build outcomes. They also leveraged the TravisTorrent dataset but restricted the study to the top ten projects with the highest number of builds.

In addition to the issue of build failures, there are several other challenges. The need to restart builds has been highlighted as a substantial drain on CI build time [50, 15, 72], and longer build durations [28, 22, 24, 27] are known to not only overuse CI resources but also increase the time to feedback. To that end, previous work also focused on reducing the time-to-feedback of CI builds [2, 3, 68, 23]. For example, Abdalkareem et al. [2, 3] proposed a method to detect when to skip CI builds based on commits that do not affect the source code (e.g., `Readme` files). This method could reduce the number of commits triggering CI by 18%.

The aforementioned prior work focused on CI builds in general. Our study thoroughly analyzes CI builds associated with commits on DevOps artefacts (i.e., DevOps builds). Besides, our study discusses how the code ownership of DevOps artefacts impacts the outcome of DevOps CI builds.

5.3 Code Ownership

Several studies have studied the relationship between code ownership and software quality [8, 81, 20, 32, 61, 19]. Bird et al. [8] conducted an empirical study on two large proprietary software projects, investigating the relationship between different code ownership metrics and the prevalence of software failures. They find that in all cases, metrics like the number of low-expertise developers and the proportion of ownership of the top owner have a relationship with the occurrence of pre-release faults and post-release failures. Foucault et al. [20] replicated the Bird et al.'s [8] study in the context of OSS projects. They studied seven OSS projects and found that the results in the OSS context were inconsistent with those of Bird et al., i.e., code ownership does not impact software quality in OSS projects. Thongtanunam et al. [81] extended the work done by Bird et al. [8] and Foucault et al. [8] by using new code ownership metrics that consider code review. They performed empirical analyses on six releases of two large OSS projects, and found that code ownership metrics that consider code review are related to software quality.

Several prior studies have delved into the dynamics of having dedicated DevOps developers versus employing full-stack developers. For example, Wiedemann et al. [87] underscore the significance of establishing a dedicated DevOps team within an organization. The study emphasizes that the expertise needed to oversee and automate IT infrastructure operations differs substantially from that of a software developer. In contrast, major tech companies, such as Amazon, advocate for a model where all software developers responsible for product development also run and operate the product. The article "Why You Should Run What You Build" by Stephen [78] highlights that developers embracing this approach apply the same creativity used in building applications to DevOps, ultimately enhancing the overall user experience.

While code ownership has been studied in the context of open source and code quality, the existing studies does not investigate the code ownership in the context of CI. Our study extends the existing studies by focusing solely on the influence of code ownership (i.e., chronological code ownership and skewness of DevOps contributions) on the DevOps CI builds.

Chapter 6

Conclusion

By studying the impact of code ownership of DevOps artefacts on the outcome and success rate of 892,193 DevOps CI builds across 1,689 OSS GitHub projects, we make the following two conclusions:

Large software organizations are better off having dedicated DevOps developers handle all DevOps-related commits. In Section 3, both RQs point to the same conclusion: higher levels of build-level (Observation 1) and project-level (Observation 4) code ownership are correlated to more successful outcomes for DevOps CI builds. Thus, we advise software practitioners to take advantage of this relationship by ensuring dedicated developers handle DevOps-related tasks in the project. However, it is essential to note that our recommendation does not advocate for an exclusive focus on DevOps responsibilities as our study did not focus on contributions to *only* DevOps. Rather, our findings underscore the importance of having individuals primarily responsible for managing DevOps artifacts within the repository. These dedicated DevOps developers can also actively engage in code development when needed, promoting a flexible and collaborative approach. Doing so allows developers with high ownership to focus on DevOps files while gaining project-specific DevOps knowledge that can benefit the project in the long run.

Smaller software organizations would benefit from all their developers working on DevOps code. Our findings also show that while having a dedicated team is paramount for larger organizations, relatively smaller organizations (e.g., six developers, as we observe in Discussion 3 in Section 3.2) could also benefit by letting all the developers work on DevOps artefacts. This collaborative approach may foster a more integrated and agile development process in the early stages of larger projects as well.

While our results show that DevOps code ownership impacts DevOps CI builds, we also propose a broader hypothesis—code ownership, in a general sense, could impact CI builds. For example, developers are often rotated among different project components, which may change their ownership of other components and ultimately affect the CI build outcomes. To the best of our knowledge, no existing work on build prediction models has considered the impact of code ownership when creating models for CI build outcome prediction. We encourage future researchers to consider the chronological code ownership and the skewness of contributions (to specific files and/or code components) when curating features for their prediction models.

Data Availability. https://zenodo.org/records/10146006

References

[1] X. Contributions to the mathematical theory of evolution.—II. Skew variation in homogeneous material. *Philosophical Transactions of the Royal Society of London. (A.)*, 186:343–414, 12 1895.

[2] Rabe Abdalkareem, Suhaib Mujahid, and Emad Shihab. A Machine Learning Approach to Improve the Detection of CI Skip Commits. *IEEE Transactions on Software Engineering*, 47(12):2740–2754, 12 2021.

[3] Rabe Abdalkareem, Suhaib Mujahid, Emad Shihab, and Juergen Rilling. Which Commits Can Be CI Skipped? *IEEE Transactions on Software Engineering*, 47(3):448–463, 3 2021.

[4] Roozbeh Aghili, Heng Li, and Foutse Khomh. Studying the characteristics of AIOps projects on GitHub. *Empirical Software Engineering*, 28(6):143, 11 2023.

[5] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 12 1974.

[6] Olusegun Akinwande, H G Dikko, and Samson Agboola. Variance Inflation Factor: As a Condition for the Inclusion of Suppressor Variable(s) in Regression Analysis. *Open Journal of Statistics*, 05:754–767, 1 2015.

[7] Eman Abdullah AlOmar, Moataz Chouchen, Mohamed Wiem Mkaouer, and Ali Ouni. Code Review Practices for Refactoring Changes: An Empirical Study on OpenStack. 3 2022.

[8] Christian Bird, Nachiappan Nagappan, Brendan Murphy, Harald Gall, and Premkumar Devanbu. Don't touch my code! In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 4–14, New York, NY, USA, 9 2011. ACM.

[9] Benjamin M Bolker, Mollie E Brooks, Connie J Clark, Shane W Geange, John R Poulsen, M Henry H Stevens, and Jada-Simone S White. Generalized linear mixed models: a practical guide for ecology and evolution. *Trends in Ecology & Evolution*, 24(3):127–135, 2009.

[10] 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.

[11] Jacob Cohen. Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. *Psychological Bulletin*, 70(4):213–220, 1968.

[12] Breno B. Nicolau de França, Helvio Jeronimo, and Guilherme Horta Travassos. Characterizing DevOps by Hearing Multiple Voices. In *Proceedings of the XXX Brazilian Symposium on Software Engineering*, pages 53–62, New York, NY, USA, 9 2016. ACM.

[13] 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, 8 2016.

[14] Paul Donner. Enhanced self-citation detection by fuzzy author name matching and complementary error estimates. *Journal of the American Society for Information Science and Technology*, 67(3):662, 2016.

[15] 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.

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

[17] Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. DevOps. *IEEE Software*, 33(3):94–100, 5 2016.

[18] Floris Erich. DevOps is Simply Interaction Between Development and Operations. pages 89–99. 2019.

[19] Csaba Faragó, Péter Hegedűs, and Rudolf Ferenc. Code ownership: Impact on maintainability. In *Computational Science and Its Applications–ICCSA 2015: 15th International Conference, Banff, AB, Canada, June 22-25, 2015, Proceedings, Part V 15*, pages 3–19. Springer, 2015.

[20] Matthieu Foucault, Jean-Rémy Falleri, and Xavier Blanc. Code ownership in open-source software. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, pages 1–9, New York, NY, USA, 5 2014. ACM.

[21] Tanner Fry, Tapajit Dey, Andrey Karnauch, and Audris Mockus. A Dataset and an Approach for Identity Resolution of 38 Million Author IDs extracted from 2B Git Commits. In *Proceedings of the 17th International Conference on Mining Software Repositories*, pages 518–522, New York, NY, USA, 6 2020. ACM.

[22] Keheliya Gallaba, John Ewart, Yves Junqueira, and Shane Mcintosh. Accelerating continuous integration by caching environments and inferring dependencies. *Transactions on Software Engineering*, 2020.

[23] Keheliya Gallaba, John Ewart, Yves Junqueira, and Shane McIntosh. Accelerating Continuous Integration by Caching Environments and Inferring Dependencies. *IEEE Transactions on Software Engineering*, 48(6):2040–2052, 6 2022.

[24] Keheliya Gallaba, Maxime Lamothe, and Shane McIntosh. Lessons from eight years of operational data from a continuous integration service. In *Proceedings of the 44th International Conference on Software Engineering*, pages 1330–1342, New York, NY, USA, 5 2022. ACM.

[25] 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 ACM/IEEE International Conference on Automated Software Engineering*, pages 87–97, 2018.

[26] Keheliya Gallaba, Martin Pinzger, Christian Macho, and Shane McIntosh. Noise and heterogeneity in historical build data: An empirical study of Travis CI. In *ASE 2018 - Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 87–97. Association for Computing Machinery, Inc, 9 2018.

[27] Taher A Ghaleb, Safwat Hassan, and Ying Zou. Studying the interplay between the durations and breakages of continuous integration builds. *Transactions on Software Engineering*, 49, 2022.

[28] 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.

[29] Georgios Gousios and Andy Zaidman. A dataset for pull-based development research. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 368–371, New York, NY, USA, 5 2014. ACM.

[30] Rubén Grande, Aurora Vizcaíno, and Félix O. García. Is it worth adopting DevOps practices in Global Software Engineering? Possible challenges and benefits. *Computer Standards & Interfaces*, 87:103767, 1 2024.

[31] Rubén Grande, Aurora Vizcaíno, and Félix O. García. Is it worth adopting DevOps practices in Global Software Engineering? Possible challenges and benefits. *Computer Standards & Interfaces*, 87:103767, 1 2024.

[32] Michaela Greiler, Kim Herzig, and Jacek Czerwonka. Code Ownership and Software Quality: A Replication Study. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 2–12. IEEE, 5 2015.

[33] Foyzul Hassan and Xiaoyin Wang. Change-Aware Build Prediction Model for Stall Avoidance in Continuous Integration. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 157–162. IEEE, 11 2017.

[34] Jordan Henkel, Christian Bird, Shuvendu K Lahiri, and Thomas Reps. Learning from, Understanding, and Supporting DevOps Artifacts for Docker. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, pages 38–49, New York, NY, USA, 2020. Association for Computing Machinery.

[35] Michael Hilton, Nicholas Nelson, Danny Dig, Timothy Tunnell, Darko Marinov, et al. Continuous integration (ci) needs and wishes for developers of proprietary code. 2016.

[36] 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.

[37] Dongyang Hu, Tao Wang, Junsheng Chang, Gang Yin, and Yang Zhang. Multi-Discussing across Issues in GitHub: A Preliminary Study. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 406–415, 2018.

[38] Jez Humble and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* Addison-Wesley Professional, 1st edition, 2010.

[39] Ramtin Jabbari, Nauman bin Ali, Kai Petersen, and Binish Tanveer. What is DevOps? In *Proceedings of the Scientific Workshop Proceedings of XP2016*, pages 1–11, New York, NY, USA, 5 2016. ACM.

[40] Ramtin Jabbari, Nauman bin Ali, Kai Petersen, and Binish Tanveer. What is DevOps? A Systematic Mapping Study on Definitions and Practices. In *Proceedings of the Scientific Workshop Proceedings of XP2016*, XP '16 Workshops, New York, NY, USA, 2016. Association for Computing Machinery.

[41] Amitkumar V. Jha, Riya Teri, Subra Verma, Susmita Tarafder, Wriddhi Bhowmik, Sunil Kumar Mishra, Bhargav Appasani, Avireni Srinivasulu, and Nsengiyumva Philibert. From theory to practice: Understanding DevOps culture and mindset. *Cogent Engineering*, 10(1), 12 2023.

[42] Xianhao Jin and Francisco Servant. What Helped, and what did not? An Evaluation of the Strategies to Improve Continuous Integration. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 213–225. IEEE, 5 2021.

[43] 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*, pages 92–101, New York, NY, USA, 5 2014. ACM.

[44] Noureddine Kerzazi and Bram Adams. Who needs release and devops engineers, and why? In *Proceedings of the international workshop on continuous software evolution and delivery*, pages 77–83, 2016.

[45] Noureddine Kerzazi, Foutse Khomh, and Bram Adams. Why do automated builds break? an empirical study. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 41–50. IEEE, 2014.

[46] Yang Luo, Yangyang Zhao, Wanwangying Ma, and Lin Chen. What are the Factors Impacting Build Breakage? In *2017 14th Web Information Systems and Applications Conference (WISA)*, pages 139–142. IEEE, 11 2017.

[47] Yang Luo, Yangyang Zhao, Wanwangying Ma, and Lin Chen. What are the Factors Impacting Build Breakage? In *2017 14th Web Information Systems and Applications Conference (WISA)*, pages 139–142. IEEE, 11 2017.

[48] H. Läuter. Silverman, b. w.: Density estimation for statistics and data analysis. chapman hall, london – new york 1986, 175 pp., £12.—. *Biometrical Journal*, 30:876–877, 1 1988.

[49] Christian Macho, Shane McIntosh, and Martin Pinzger. Automatically repairing dependency-related build breakage. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 106–117. IEEE, 2018.

[50] 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.

[51] 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(5):2146–2189, 10 2016.

[52] Ade Miller. A hundred days of continuous integration. In *Agile 2008 conference*, pages 289–293. IEEE, 2008.

[53] Sikender Mohsienuddin Mohammad. Devops automation and agile methodology. *International Journal of Creative Research Thoughts (IJCRT), ISSN*, pages 2320–2882, 2017.

[54] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. Curating GitHub for engineered software projects. *Empirical Software Engineering*, 22(6):3219–3253, 12 2017.

[55] Reza Nadri, Gema Rodriguez-Perez, and Meiyappan Nagappan. On the Relationship Between the Developer's Perceptible Race and Ethnicity and the Evaluation of Contributions in OSS. *IEEE Transactions on Software Engineering*, 48(8):2955–2968, 8 2022.

[56] Shinichi Nakagawa and Holger Schielzeth. A general and simple method for obtaining R2 from generalized linear mixed-effects models. *Methods in Ecology and Evolution*, 4(2):133–142, 2 2013.

[57] Ansong Ni and Ming Li. Cost-Effective Build Outcome Prediction Using Cascaded Classifiers. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 455–458. IEEE, 5 2017.

[58] Ansong Ni and Ming Li. Cost-Effective Build Outcome Prediction Using Cascaded Classifiers. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 455–458. IEEE, 5 2017.

[59] Zi Peng, Tse-Hsun Chen, and Jinqiu Yang. Revisiting test impact analysis in continuous testing from the perspective of code dependencies. *IEEE Transactions on Software Engineering*, 48(6):1979–1993, 2020.

[60] Jackson A Prado Lima, Willian DF Mendonça, Silvia R Vergilio, and Wesley KG Assunção. Cost-effective learning-based strategies for test case prioritization in continuous integration of highly-configurable software. *Empirical Software Engineering*, 27(6):133, 2022.

[61] Foyzur Rahman and Premkumar Devanbu. Ownership, experience and defects: a fine-grained study of authorship. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 491–500, 2011.

[62] Ayushi Rastogi, Nachiappan Nagappan, Georgios Gousios, and André van der Hoek. Relationship between geographical location and evaluation of developer contributions in github. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 1–8, New York, NY, USA, 10 2018. ACM.

[63] Ayushi Rastogi, Nachiappan Nagappan, Georgios Gousios, and André van der Hoek. Relationship between geographical location and evaluation of developer contributions in github. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 1–8, New York, NY, USA, 10 2018. ACM.

[64] Leah Riungu-Kalliosaari, Simo Mäkinen, Lucy Ellen Lwakatare, Juha Tiihonen, and Tomi Männistö. Devops adoption benefits and challenges in practice: A case study. In *Product-Focused Software Process Improvement: 17th International Conference, PROFES 2016, Trondheim, Norway, November 22-24, 2016, Proceedings 17*, pages 590–597. Springer, 2016.

[65] Islem Saidani, Ali Ouni, Moataz Chouchen, and Mohamed Wiem Mkaouer. Predicting continuous integration build failures using evolutionary search. *Information and Software Technology*, 128:106392, 2020.

[66] Islem Saidani, Ali Ouni, and Mohamed Wiem Mkaouer. Improving the prediction of continuous integration build failures using deep learning. *Automated Software Engineering*, 29, 2022.

[67] Islem Saidani, Ali Ouni, and Mohamed Wiem Mkaouer. Improving the prediction of continuous integration build failures using deep learning. *Automated Software Engineering*, 29(1):21, 5 2022.

[68] Mark Santolucito, Jialu Zhang, Ennan Zhai, Jurgen Cito, and Ruzica Piskac. Learning CI Configuration Correctness for Early Build Feedback. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 1006–1017. IEEE, 3 2022.

[69] Steven Sawyer. Analysis of Variance: The Fundamental Concepts. *Journal of Manual & Manipulative Therapy*, 17:27E–38E, 4 2009.

[70] Gideon Schwarz. Estimating the Dimension of a Model. *The Annals of Statistics*, 6(2):461–464, 1978.

[71] Tushar Sharma, Marios Fragkoulis, and Diomidis Spinellis. Does your configuration code smell? In *Proceedings - 13th Working Conference on Mining Software Repositories, MSR 2016*, pages 189–200. Association for Computing Machinery, Inc, 5 2016.

[72] 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.

[73] Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes? *ACM SIGSOFT Software Engineering Notes*, 30(4):1–5, 7 2005.

[74] Jens Smeds, Kristian Nybom, and Ivan Porres. DevOps: A Definition and Perceived Adoption Impediments. pages 166–177. 2015.

[75] Eliezio Soares, Daniel Alencar da Costa, and Uirá Kulesza. Continuous Integration and Software Quality: A Causal Explanatory Study. 9 2023.

[76] Eliezio Soares, Gustavo Sizilio, Jadson Santos, Daniel Alencar da Costa, and Uirá Kulesza. The effects of continuous integration on software development: a systematic literature review. *Empirical Software Engineering*, 27(3):78, 2022.

[77] Thodoris Sotiropoulos, Stefanos Chaliasos, Dimitris Mitropoulos, and Diomidis Spinellis. A model for detecting faults in build specifications. *Proceedings of the ACM on Programming Languages*, 4(OOPSLA):1–30, 11 2020.

[78] Stephen Orban. Enterprise DevOps: Why You Should Run What You Build, 2015.

[79] Ahmed Tamrawi, Hoan Anh Nguyen, Hung Viet Nguyen, and Tien N Nguyen. Build code analysis with symbolic evaluation. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 650–660. IEEE, 2012.

[80] Minaoar Hossain Tanzil, Masud Sarker, Gias Uddin, and Anindya Iqbal. A mixed method study of DevOps challenges. *Information and Software Technology*, 161:107244, 9 2023.

[81] Patanamon Thongtanunam, Shane McIntosh, Ahmed E Hassan, and Hajimu Iida. Revisiting Code Ownership and Its Relationship with Software Quality in the Scope of Modern Code Review. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 1039–1050, New York, NY, USA, 2016. Association for Computing Machinery.

[82] 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 2015 10th joint meeting on foundations of software engineering*, pages 805–816, 2015.

[83] Carmine Vassallo, Gerald Schermann, Fiorella Zampetti, Daniele Romano, Philipp Leitner, Andy Zaidman, Massimiliano Di Penta, and Sebastiano Panichella. A tale of ci build failures: An open source and a financial organization perspective. In *2017 IEEE international conference on software maintenance and evolution (ICSME)*, pages 183–193. IEEE, 2017.

[84] Carmine Vassallo, Fiorella Zampetti, Daniele Romano, Moritz Beller, Annibale Panichella, Massimiliano Di Penta, and Andy Zaidman. Continuous delivery practices in a large financial organization. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 519–528. IEEE, 2016.

[85] Mario Villamizar, Oscar Garcés, Lina Ochoa, Harold Castro, Lorena Salamanca, Mauricio Verano, Rubby Casallas, Santiago Gil, Carlos Valencia, Angee Zambrano, et al. Cost comparison of running web applications in the cloud using monolithic, microservice, and aws lambda architectures. *Service Oriented Computing and Applications*, 11:233–247, 2017.

[86] Yuqing Wang, Maaret Pyhäjärvi, and Mika V Mäntylä. Test automation process improvement in a devops team: experience report. In *2020 ieee international conference on software testing, verification and validation workshops (icstw)*, pages 314–321. IEEE, 2020.

[87] Anna Wiedemann, Manuel Wiesche, Heiko Gewald, and Helmut Krcmar. Integrating development and operations teams: A control approach for DevOps. *Information and Organization*, 33(3):100474, 9 2023.

[88] 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)*, pages 5–8. IEEE, 2019.

[89] Jing Xia and Yanhui Li. Could We Predict the Result of a Continuous Integration Build? An Empirical Study. In *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 311–315. IEEE, 7 2017.

[90] Jing Xia, Yanhui Li, and Chuanqi Wang. An Empirical Study on the Cross-Project Predictability of Continuous Integration Outcomes. In *2017 14th Web Information Systems and Applications Conference (WISA)*, pages 234–239. IEEE, 11 2017.

[91] Zheng Xie and Ming Li. Cutting the Software Building Efforts in Continuous Integration by Semi-Supervised Online AUC Optimization. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 2875–2881, California, 7 2018. International Joint Conferences on Artificial Intelligence Organization.

[92] Zheng Xie and Ming Li. Cutting the Software Building Efforts in Continuous Integration by Semi-Supervised Online AUC Optimization. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 2875–2881, California, 7 2018. International Joint Conferences on Artificial Intelligence Organization.

[93] Jiaxin Zhu, Minghui Zhou, and Audris Mockus. Patterns of Folder Use and Project Popularity: A Case Study of Github Repositories. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '14, New York, NY, USA, 2014. Association for Computing Machinery.

[94] Daniel Zwillinger and Stephen Kokoska. *CRC Standard Probability and Statistics Tables and Formulae.* CRC Press, 12 1999.