CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

The Impact of Adopting Continuous Integration on the Delivery Time of Pull Requests

A Partial Replication and Extension

Master's thesis in Computer science and engineering

YUNFANG GUO

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2019

Master's thesis 2019

The Impact of Adopting Continuous Integration on the Delivery Time of Pull Requests

A Partial Replication and Extension

YUNFANG GUO

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2019

The Impact of Adopting Continuous Integration on the Delivery Time of Pull Requests
A Partial Replication and Extension
YUNFANG GUO

Supervisor: Philipp Leitner, Department of Computer Science and Engineering
Examiner: Christian Berger, Department of Computer Science and Engineering

Master's Thesis 2019
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2019

The Impact of Adopting Continuous Integration on the Delivery Time of
Pull Requests
A Partial Replication and Extension
Yunfang Guo
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The practice of Continuous Integration (CI) is indispensable in the software development process nowadays. In a distributed and collaborative context, CI is a paradigm for integrating all changes from different developers quickly and safely. Thus the adoption of CI is considered to fasten delivering new functionalities. A study by Bernardo *et al.* doubted this claim, conducted an empirical study and found that only half of the investigated projects deliver Pull Requests (PR) faster after CI. In this study, considering the threats to the validity in their study, as well as the generalizability of their findings, a partial replication study, a comparison to projects that never apply CI, and an extension study based on other projects with CI are carried out. Through analysis on 132,006 merged PRs of 82 GitHub projects, I find that the results of Bernardo *et al.*'s work still stand, that CI could not always fasten delivery activities after CI. In addition, by comparing the statistical performance of projects that apply CI with those that never apply CI, it is proved that similar changes in development activities can only be observed in projects with CI adopted.

Keywords: continuous integration, pull-based development, replication study, delivery delay, Travis-CI

Acknowledgements

Firstly I would like to express my sincere gratitude to my supervisor Philipp Leitner, who guided me through the thesis and provided valuable opinions to the thesis work. Your constant feedback during my study and my writing helped me a lot. I would like to thank Christian Berger for being my examiner and for being so helpful throughout the whole process.

Yunfang Guo, Gothenburg, May 2019

Contents

List of Figures

List of Tables

1

Introduction

The concept of Continuous Integration (CI), which was promoted by Martin Fowler's 2000 blog post [1], is now a popular practice in software community [2]. It helps the developers integrate changes frequently in a collaborative manner [1]. As a distributed and cooperative practice, CI is commonly used in both commercial (e.g. Microsoft [3]) and open source software (OOS) development (e.g. Mozilla [4]). Though CI is more popular and important on OOS platforms (e.g. GitHub) due to the inner attributes of OOS projects like contributors are volunteers and geographically distributed [5].

Much previous research has investigated the impacts brought by CI on OOS projects. Vasilescu *et al.* [6] found that the core developers discover more bugs using CI. PRs submitted by core developers are more likely to get processed, accepted and merged with CI. Ståhl and Bosch claim that the integrators tend to release more frequently after CI [7]. Different from their studies, inspired by Bernardo *et al.*'s work [8], I report on a study of how the adoption of CI impacts the delivery behavior of Pull Requests (PR) in GitHub open source projects in this paper.

1.1 Statement of the Problem

A recent publication by Bernardo *et al.* [8] did interesting research in this field. Their work empirically analyzed whether CI improves the time-to-delivery of merged Pull-Requests (PRs) that are submitted to GitHub projects. The results revealed that only 51.3% of the projects deliver merged PRs more quickly after adopting CI. The authors claimed that the large increase of PR submission, merge and delivery rate after CI was a possible reason as to why projects deliver PRs more slowly after the CI introduction. However, for Research Question 1 (RQ1) and RQ2 in this paper, the authors have chosen to compare the overall statistics of before and after applying CI, which may have considerable threats. Firstly, they did not consider whether hidden patterns exist in regard to time, for example a growth in the number of submitted PRs, which may falsify the conclusion. Secondly, they did not compare their results with projects that never applied CI in the first place.

Specifically, the most crucial internal threat might exist in the approach of observing the pattern of the merge delay—the time interval between a PR was created and merged—and the delivery delay—the time interval between when it's merged and released. They analyzed the two metrics by testing the null hypothesis that is two distributions come from the same population ($\alpha = 0.05$). The threat could be that the number of PR submission, merge and delivery may have a relationship with time and normal growth in pull request submissions. In the thesis work, efforts are made to investigate if such hidden pattern exists and whether the results of the authors still hold if the pattern is controlled for.

Another important threat is that their control was limited to projects that applied CI, and did not take projects which never applied CI into consideration. If one would find similar results even for projects that never integrated CI, then CI cannot be the causal reason for the results the original paper observed. Thus, control over projects employed CI and never employed CI is conducted in this thesis.

One possible external threat would be generalizability. In the replication part of this study, data is collected from a number of different open source projects in GitHub. The analysis is carried out using the same approaches and verify if the results still hold for new data.

Hence, the work of this thesis project includes (1) conducting a partial replication study of the RQ1 and RQ2 of the original paper, (2) validating their findings by applying another methodology that I consider more robust with regarding to the above-mentioned internal threats, (3) supporting the generalizability through collecting and analyzing more open source projects, as well as (4) extending their study to get deeper understanding on the delivery process of a PR. In RQ3 of Bernardo *et al.*'s work, they investigated which factors have a strong influence on the delivery time of PRs after adopting CI by using multiple regression modeling. In this thesis, new metrics are added to the model, evaluated and compared to the previous metrics.

1.2 Research Questions

This study investigated the following Research Questions (RQs):

RQ0: *Can the original results be reproduced?* The first task of this study is a partial replication of the original paper. The necessity of conducting replication study is provided by many previous works as stated in Section 3.4, and rooted from the authors' concern for the internal threats to the validity. Besides, replication works can give me a more detailed, complete understanding of the original study, which lays a good foundation for later research tasks.

RQ1: *Can similar results to the original study are found when applying an alter-*

native statistical method? This question is raised because of the concern for the impact of normal growth in PR submissions on the results. It is investigated using the same projects and data, with a different methodology. The results show that consistent with the original study, CI adoption does not lead to a faster delivery speed of merged PRs.

RQ2: *Are there any other metrics that have strong explanatory power if added to the linear regression model in the original paper?* To investigate what other metrics could explain the delivery time of a PR, a new metric which focuses on the time referring to when a PR came into the release cycle shows strong influence in some of the models.

RQ3: *Can similar results be found when applying the same methodology to projects that never introduced CI?* This question is put forward because of the concern for the results of controlling over projects that never employed CI. The outcome indicates that the performance of projects without CI is more different than the same as the original findings.

RQ4: *Can similar results are found when applying the same methodology to different projects that introduced CI in their lifecycle?* This research question aims to test the generalizability of the original study, and the result reveals that the new projects share similar statistical performance with the projects in the original study.

1.3 Outline

Chapter 2 identifies the key concepts used throughout this study. The last section of this chapter summarizes the approaches and main results from the original study, to give the reader an overview of what this paper is replicating. Chapter 3 presents previous related work, as well as the necessity of replication work. Chapter 4 describes the way of collecting projects and metadata, and the research methodologies used. The results from this study along with a comparison with the original paper are presented in Chapter 5. Chapter 6 discusses the study's threats to validity. Lastly, this paper ends with a summary of the thesis work and suggested future work in Chapter 7.

2

Background

This chapter explains the important concepts that a reader would need to know to understand this work. Background knowledge of Continuous Integration, GitHub and Pull-based Development are the core concepts in this study. Besides, an overview of the original study is presented in Section 2.3.

2.1 Continuous Integration

Continuous Integration (CI) is a practice which has originated from Agile software development and Extreme Programming. It is a development practice of merging all developer working copies to shared mainline several times a day. Each integration is then verified by an automated build, which allows the errors to be detected and located as early as possible [9]. CI is intended to bring many benefits, such as quickening the delivery of new functionalities [10], reducing problems of code integration in a collaborative environment [11] and therefore the stability of the code in the main repository is guaranteed. Based on the above reasons, CI is widely used in practice nowadays. In particular, the popular projects favor CI according to the study of Hilton *et al.* [12]. Some well-known CI services are Jenkins, Travis-CI, Bamboo, TeamCity, etc.

2.2 GitHub and Pull-based Development

There are two general ways in which people contribute to open source projects on GitHub, shared repository and pull-based development. In the way of a shared repository, the owner of the repository shares read and write access to the main repository, enabling external contributors to push their local changes directly to the central repository. The study focuses on pull-based development whose development process is shown in Figure 2.1. The detailed process is described below.

Step1: The main repository is not shared with the external developers. Instead, the

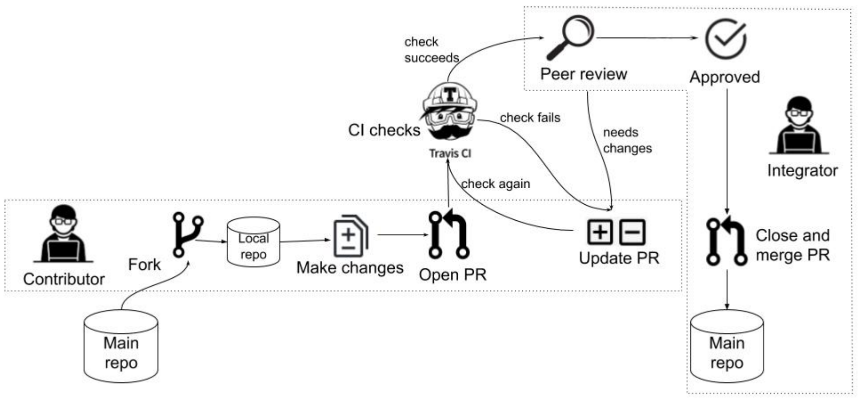


Figure 2.1: An overview of the Pull-based Development on GitHub

contributor forks the central repository then clones it to a local repository. Now the owner of the forked repository is the contributor, so any changes can be made to this repository without inferring the main project or other forked repositories.

Step2: The contributor makes changes to the local repository, commits the changes.

Step3: The contributor submits the local changes to the main repository by opening a Pull Request in the central repository. This action is either done on the GitHub portal, or by GitHub API, or by git commands. The opened PR specifies which branch that the changes will be merged into.

Step4: After a PR is submitted, the Travis-CI service automatically merges it into a test branch. The service then runs all the tests for the whole project to check if the PR breaks the codebase. If the PR doesn't pass the CI check, the contributor will need to update the PR until it passes the check.

Step5: After careful inspection to the changes in the PR, and several rounds of discussion regarding how to improve the code quality or new ideas about the functionalities, the integrator decides to approve the PR, then merges and closes it. A PR could be closed without merged either by an integrator or the author of the PR.

2.3 Summary of the Original Study

As the work of this study is a partial replication and extension to the work of Bernardo et al[8], a quick summarization of their study is necessary. To investigate whether CI improves the delivery time of merged Pull-Requests in open source projects on GitHub, they collected 162,653 PRs and 7,440 releases of 87 projects by GitHub API. They addressed the following three research questions:

RQ1: Are merged pull requests released more quickly using continuous integration?
RQ2: Does the increased development activity after adopting CI increases the delivery time of pull requests?
RQ3: What factors impact the delivery time after adopting continuous integration?

By applying non-parametric tests to the merge time, delivery time of PRs, they drew the conclusion for RQ1 that only half of the projects deliver merged PRs faster after adopting CI. And surprisingly, 71.3% of the studied projects merge PRs faster before CI, not after CI. In RQ2, through non-parametric tests, they deducted that the considerable increase on PR submission, merge and delivery rate and the sum of code churn per release could be possible reasons as to why projects do not deliver merged PRs faster after adopting CI. Besides, the number of releases per year does not change significantly after CI. In RQ3, they built linear regression models for each project and used the Wald X^2 maximum likelihood test to evaluate the explanatory power of each variable. The result showed that after CI adoption, a PR had a smaller delivery time when the time it was merged was closer to the end of the release cycle.

3

Related Work

This chapter describes the previous work in related fields and how the research questions in the study try to fill in gaps presented in the field. Section 3.1 presents findings in applying pull-request-based development in open source projects. Section 3.2 describes the influence of the usage of CI from varies aspects. Following this, Section 3.3 revolves around the effects caused by the interaction between CI and PR, which is also the research focus of this study. Finally, Section 3.4 shows the necessity of conducting replication study and the present situation of a replication study in the Software Engineering field.

3.1 Pull-request-based Development

Related works have shown that PR-based-development has its advantages in open source projects. Vasilescu *et al.* [11] collected 223 GitHub projects and found that for an overwhelming majority of projects (39 out of 45, or 87%), builds corresponding to PRs are much more likely to succeed than builds corresponding to direct pushes. Gousios *et al.* [13] found that 14% of repositories are using PRs on GitHub. They selected 291 projects from GHTorrent corpus and conducted empirical researches. The conclusion is the PR model offers fast turnaround, increased opportunities for community engagement and decreased time to incorporate contributions. They also investigated the lifecycle characteristics of PRs and determining the factors that affect them. The factors including the size of PR, tests, discussion, and code review. Different from Gousios *et al.*, I focus on the quantitative impact of the introduction of CI in PR's delivery time.

3.2 How Adopting CI Influences Projects

Previous researchers have investigated the impact of adopting CI in projects from multiple aspects. Most papers showed that the introduction of CI is beneficial to projects. Manglaviti *et al.* [14] examined the human resources that are associated with developing and maintaining CI systems. They analyzed 1,279 GitHub repositories that adopt TRAVIS CI using quantitative methods. The authors found that for projects with growing contributor bases, adopting CI becomes increasingly beneficial and sustainable as the projects ages.

There is a strong expectation that CI should improve the productivity of projects. Ade Miller [15] analyzed influence made by CI by summarizing their experience with CI in a distributed team environment at Microsoft in 2007. He collected various data related to CI in their daily work then did quantitative analysis. He claimed that teams moving to a CI driven process can expect to achieve at least a 40% reduction in check-in overhead when compared to a check-in process that maintains the same level of the code base and product quality. Ståhl and Bosch [7], argued based on survey-based evidence [16] that build and test automation saves programmers time for more creative work, and should thus increase productivity. Stolberg argued that CI practices could speed up the delivery of software by decreasing integration times [17]. However, not all researches agree on the adoption of CI improves productivity. For instance, Parsons *et al.* [18] find no clear benefits of CI on either productivity or quality.

3.3 The Interaction Between CI and PRs

This paper focuses on whether CI has an impact on the PRs and there are already many previous works in this research track. Hilton *et al.* [19] analyzed 34,544 open source projects from GitHub and surveyed 442 developers. The authors found that 70% of the most popular GitHub projects use CI. The authors found that CI helps projects release twice as often and that when using CI, the PRs are accepted 1.6 hours sooner in median.

Vasilescu *et al.* [11] studied the usage of Travis-CI in a sample of 223 GitHub projects written in Ruby, Python, and Java. They found that the majority of projects (92.3%) are configured to use Travis-CI, but less than half actually use it. In follow-up research, Vasilescu *et al.* [6] investigated the productivity and quality of 246 GitHub projects that use CI. They found that projects that use CI could have more PRs got processed, more are being accepted and merged when they are submitted by core developers. This increased productivity does not appear to be gained at the expense of quality.

Zhao *et al.* [20] conducted an empirical study to investigate the transition to Travis-CI in a large sample of GitHub open-source projects. They quantitatively compared the CI transition in these projects using metrics such as commit frequency, code churn, PR closing, and issue closing. In addition, they conducted a survey with a sample of developers of those projects. They used a set of three questions related to the adoption of Travis and CI. They also asked how their development process was adapted to accommodate the transition to CI. The main results of their study are: (i) a small increase in the number of merged commits after CI adoption; (ii) a statistically significant decreasing in the number of merged commit churn; (iii) a moderate increase in the number of issues closed after CI adoption; and (iv) a stationary behavior in the number of closed PRs as well as a longer time to close PRs after the CI Adoption.

Yu *et al.* [21] collected 103,284 PRs from 40 different GitHub projects. They tried to investigate which factors affect PR evaluation latency in GitHub by applying a linear regression model and quantitative analysis. They found that the size of PR matters and the availability of the CI pipeline are actional strong predictors. In Yu *et al.*'s later work [22], they again used a linear regression model to analyze which factors affect the process of the pull-based development model in the context of CI. The authors found that the likelihood of rejection of a PR increase by 89.6% when the PR breaks the build. The results also show that the more succinct a PR is, the greater the probability that such a PR is reviewed and merged earlier.

3.4 Replications Studies

As Bernardo *et al.* suggested, replications of their study in different settings are necessary [8]. The necessity of replication study has been supported by many prior works. The results of any one study cannot simply be extrapolated to all environments because there are many uncontrollable sources of variation between different environments [23]. Successful replication increases the validity and reliability of the outcomes observed in an experiment [24]. There are studies in criminology and education area reporting that replication studies are necessary, but too rare [25][26]. More generally, in the social and life sciences, researchers have realized that the replication crisis is an ongoing methodological crisis since the results of many scientific studies are difficult or impossible to replicate or reproduce on subsequent investigation, either by independent researchers or even by the original researchers themselves [27].

Basili *et al.* agreed and claimed that replication study in the software engineering is also necessary but difficult [28]. It's because an experiment in this field usually involve a large number of context variables. However, according to the study of Silva *et al.* [29] by presenting a systematic mapping study of replications in the software engineering field, they concluded that the absolute number of replications is still small, in particular considering the breadth of topics in software engineering. Their study retrieved more than 16,000 articles, from which they selected 96 articles reporting 133 replications. In further detailed, an empirical study in the Agile Release Engineering (ARE) field is necessary according to Karvonen *et al.* [30]. Through a systematic literature review, they analyzed 619 papers and selected 71 primary studies that are related to ARE practices. They found that only 8 out of the 71 primary studies empirically investigate CI. They highlighted that empirical research in this field is highly necessary to better understand the impact of adopting

CI on software development. Due to these reasons, in this report I replicate part of the original research to validate the original results and conduct some extension to further explore the studied area.

4
Approach

This chapter describes the way of collecting projects and metadata in Section 4.1. In Section 4.2, the methodologies for analyzing the data are detailed. Figure 4.1 gives an overview of the dataset that is used into different RQs respectively and distinct methodologies applied in each RQ. For RQ0, RQ1, and RQ2, the dataset from the original study with additional data is applied. The RQ3 and RQ4 use an new dataset collected from GitHub. For the first three RQs, the methods and procedures for different analysis purposes are reused from the original study as well as inspired by related previous works. The last two RQs apply all analysis methods in the first three RQs.

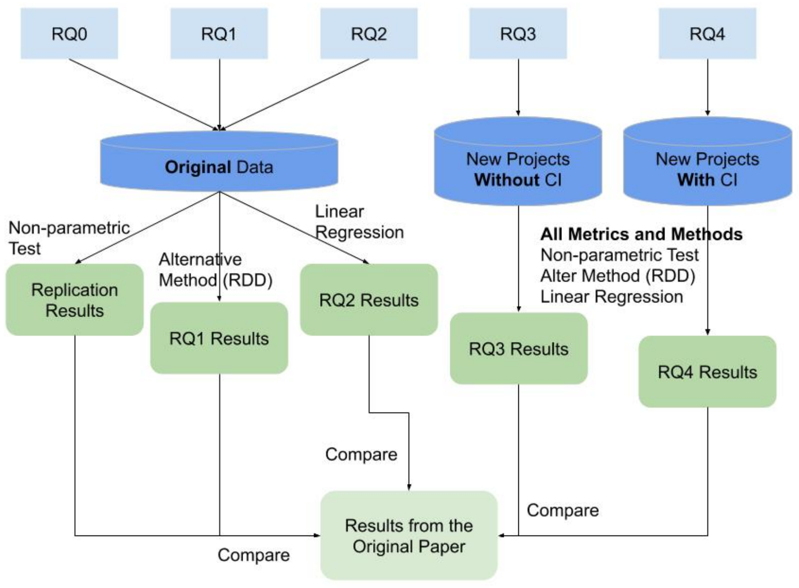


Figure 4.1: The data fed into RQs, the methods applied in each RQ.

4.1 Data Collection

Section 4.1.1 describes how the new projects used in this study were collected. Section 4.1.2 shows how the metadata of PRs and releases was collected.

4.1.1 Studied Projects

The 87 previous projects are investigated in RQ0–the original study and the metadata of the projects have been published by the authors online[1]. RQ1 and RQ2, which are considered validation and extension to the original study, need additional data such as the exact time point when a PR was merged for the 87 projects. This information is collected directly through the GitHub API.

For the research work in RQ3 and RQ4, two new datasets are required. Projects with CI adoption will be used in RQ4, while projects that never apply CI are required in RQ3. The process of project selection closely follows the process in the original study [8]. In Figure 4.2, the upper workflow depicts the process of collecting projects with CI adoption and the bottom workflow shows the procedure of collecting another set of projects.

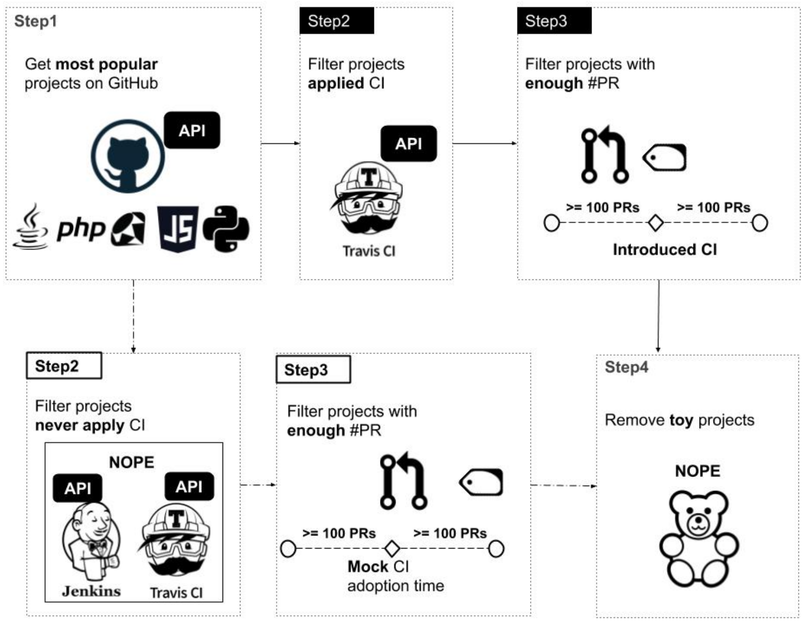


Figure 4.2: The process of project selection. The upper workflow shows the process of collecting projects with CI, the bottom workflow shows the procedure of collecting projects without CI.

Step1: identify the 800 most popular projects for each of these languages. Java, Python, PHP, Ruby, and Javascript on GitHub making use of GitHub API. The

[1]https://prdeliverydelay.github.io/#datasets

popularity depends on the number of stars of a project. This search was done on April 5th, 2019. There are 2763 projects in all. The reason for the total number is 2763 instead of 4000 is if one project using both Java and Javascript was collected twice. Besides the projects had been used in the original study are filtered out for the analysis purpose of validating the original results by using completely different datasets.

Step2: check whether a project applied CI tools. In this study, only projects adopted Travis-CI are considered valid projects with CI. Projects that use Jenkins don't keep the entire CI history, thus there is no way of knowing the time a project introduced CI. The date of the first Travis-CI build is collected using the Travis-CI API, and the time of when the first build was created is considered the CI adoption time. 1652 out of 2763 projects adopted Travis-CI and 1,111 do not apply Travis-CI.

Step3: gather merged PRs of each project. Similar to Bernardo *et al.*, this study excludes projects that have less than 100 merged PRs before or after the time when CI was adopted, so as to guarantee enough data to conduct later analysis. A similar timepoint is chosen for projects do not apply CI by an investigation from the originally studied 87 projects. Dividing the days before the CI introduction over the overall analyzed lifetime for every project, I get a mean of 0.382, a median of 0.38, a variance of 0.085. In other words, from the sample of 87 projects, I could conclude that projects usually introduces CI service after 38% of its lifetime. Then I define the timepoint where 38% of its lifetime as the "mock-ci-timepoint" for projects without CI adoption. Again, projects that have less than 100 merged PRs before or after the mock-ci-point are excluded. Firstly by querying the MongoDB raw dataset of the project GHTorrent, which mirrors data offered by GitHub Restful API[2] [31], I get the number of PRs of each project, then filter out the projects with less than 200 PRs. Only 103 remain among 1,111 projects without CI adoption, 295 remain among 1652 projects that applied CI projects. Furthermore, after linking PRs to releases (the first two steps in Section 4.1.2), projects without releases before or after CI are removed. Projects have less than 100 linked PRs before or after CI adoption are also excluded, and the same is true for projects without CI adoption. 28 out of 98 no-ci-projects are left, 54 out of 295 ci-projects remain.

Step4: filter out toy projects. GitHub is an open platform that hosts a variety of projects. A toy project could be a student's homework or for different purposes other than general software development activities, for example, this **kelseyhightower/nocode**[3] is incredibly popular but was just created for fun. In a case similar projects remain after the former three steps, they should be excluded so as to keep functional-similar projects and ensure the accuracy of this study.

[2]`https://developer.github.com/`

[3]`https://github.com/kelseyhightower/nocode`

4.1.2 Data Collection Process

The data collection process is partially parallel to the project selection process. Figure 4.3 shows the process of collecting PR and release information for every project, no matter if it adopts CI or not. Each step of the process is detailed below.

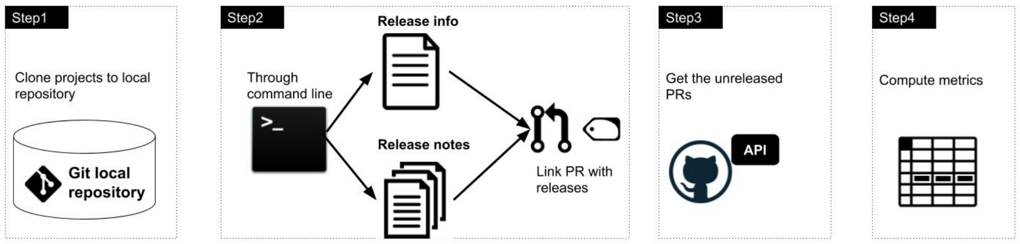


Figure 4.3: The process of collecting data.

Step1: clone projects to local repository. Although almost information of a project is available via the GitHub API, sometimes it is more efficient to clone the project locally. In this study the data of releases and tags are needed. It is more convenient to gather the releases and related information via a git command in a local project folder.

Step2: link PRs to releases respectively. Firstly all releases of a project are collected via git command along with with start date, publish date and duration of a release. On GitHub one can make a *pre-release* which is identified as non-production ready[4]. It is found that for one production-intended release, there might be more than one pre-release, and usually a pre-release is named with a prefix or suffix pre, beta, rc (release candidate), alpha. In this study, the pre-releases are filtered out to decrease the variety in data because the integrator may launch pre-releases arbitrarily. Then I compute the diff between two consecutive releases, and the diff lists commit that were included in one release but not in another. In the commits log, I locate the record of merging a PR by searching for commits with message 'Merge pull request #<X>' (which is automatically generated by git when a PR is merged). For example, a commit with message 'Merge pull request #312' is found in the diff between v1.0 and v1.1, v1.0 is released earlier than v1.1, then I link the #312 PR to the release v1.0. After a merged and delivered PR is found, I send a request via GitHub API to collect the information of a PR: *create date, merge date, close date, latest update date, PR id, author, the number of added and deleted lines (churn), pull number.*

Step3 and Step4: get the unreleased PRs by GitHub API then compute metrics on the ground of all metadata. Apart from the merged and released PRs, metadata of merged and unreleased PRs are necessary to the analysis work. There are 67,309 released PRs and 17,178 merged but unreleased PRs from 54 projects that adopted CI, 35,123 released as well as 12,396 merged but not released PRs from 28 projects

[4]`https://help.github.com/en/articles/creating-releases`

without CI adoption. The detailed description of PR-wise and release-wise metrics are covered in Section 5.3.

4.2 Analysis Work

For the different analysis purposes of each research question, the collected data and computed metrics are analyzed by different methodologies as described in this section.

4.2.1 Non-parametric tests

The first part of the thesis work is a partial replication of the RQ1 and RQ2 of the original study. The two research questions focused on the statistical performance of the metrics the authors proposed by comparing the data distribution before and after the CI adoption, that if the performance of one metric changes with a high significance, then one can claim that the introduction of CI strongly influenced the projects from this perspective. The authors applied non-parametric tests and got good results, hence the methods are again used in the analysis work of RQ3 and RQ4 in this study.

In further detailed, for each metric, the data is split into two buckets regarding whether the time data of the record was before the CI adoption or after. Firstly the authors applied the Mann-Whitney-Wilcoxon (MWW) test, which is a non-parametric test that does not require the assumption of normal distributions. The null hypothesis is two distributions come from the same population with $\alpha = 0.05$. If a result of a *p-value* is smaller than the α, the null hypothesis is rejected in favor of the alternative hypothesis that the two distributions come from different populations. Speaking of the analyzed metrics, different distributions are expected because the authors assumed the adoption of CI fastens the Pull-based development. Then another non-parametric metric–Cliff's delta was used to measure the magnitude of the difference between the two distributions. The higher Cliff's delta value, the greater the difference between distributions. Generally, the standards set by Romano *et al.* [27] are used, that is a delta < 0.147 is considered negligible difference, a delta < 0.33 shows small difference, a delta < 0.474 is considered medium and a delta $>= 0.474$ is seen as large difference. Likewise, large Cliff's delta values are expected.

4.2.2 Regression Discontinuity Design

4.2.2.1 Motivation

RQ1 was proposed due to the concern of the impact of hidden patterns regarding time series in metrics, e.g. a normal growth in PR submissions on the results. The choice of Regression Discontinuity Design (RDD) is inspired by the research of Zhao *et al.* [20]. Since the RDD is the most powerful among varies quasi-experimental designs to asses the existence and extent of the impact introduced by an intervention [32], in the study of Zhao *et al.*, RDD was applied to measure the longitudinal effect made by the adoption of Travis-CI to their analyzed projects. The results gave them a graphical overview of the trend before and after the intervention as well as significant results through statistical analysis. The effectiveness and convenience of the RDD motivate me to apply it in this study.

4.2.2.2 Identification

RDD is a fairly old idea firstly proposed by Thistlethwaite and Campbell at 1960 [33]. It experiences a renaissance in recent years [34]. It is a quasi-experimental pretest-posttest design that elicits the causal effects of interventions by assigning a cutoff above or below which an intervention is assigned. The assumption of RDD is that the trend continues without changes if the intervention does not exist. In this case, the intervention refers to the adoption of CI, a cutoff refers to the time point when CI was introduced. The expected result in this study is an obvious discontinuity around the cutoff point for the analyzed metric.

4.2.2.3 Estimation

In this study, several models are chosen to fit the data. Firstly define variables that appear in the models.

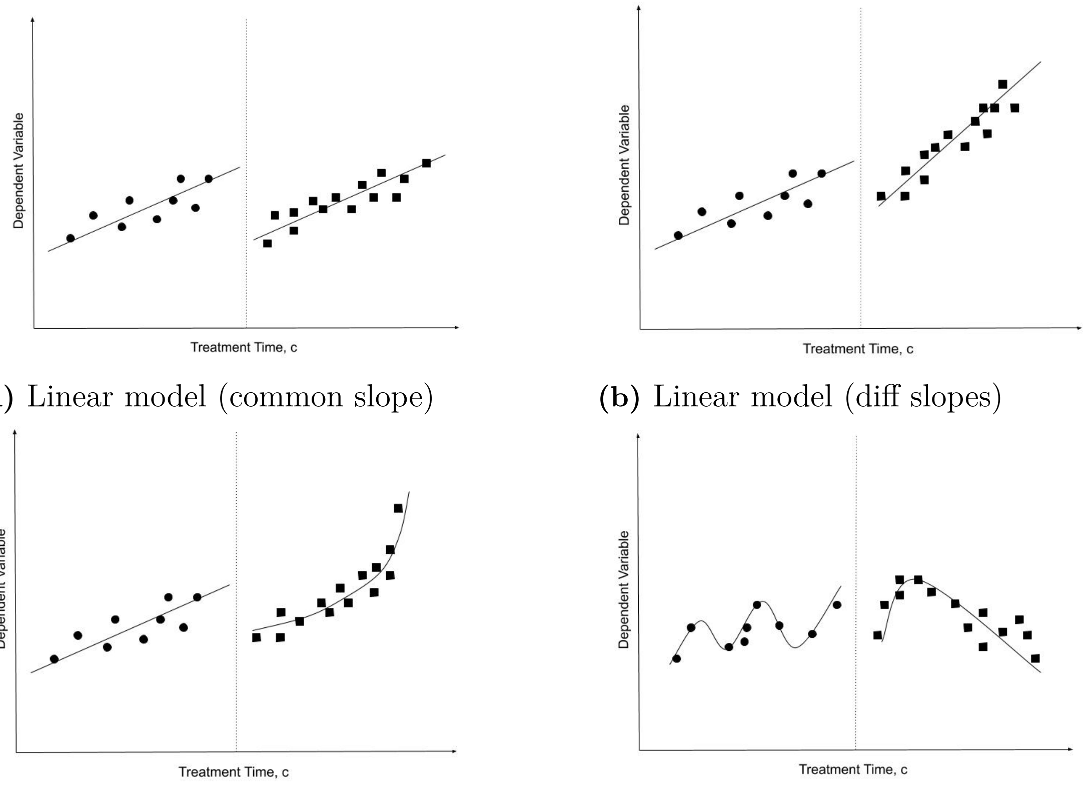
- $D_i \in 0, 1$ as the intervention, which is also called the treatment, refers to the timepoint when CI service was adopted.
- X_i as the forcing variable, the time difference between when a PR was created and when the CI was adopted in weeks. It totally determines the value of D_i with cutpoint c. $D_i = 1 \{X_i > c\}$
- $Y_i(1)$ and $Y_i(0)$ represent the value of the dependent variable (delivery delay) of the corresponding PR record with and without the impact of intervention respectively.

- $\mathbb{E}[Y_i(\mathrm{d}) \mid X_i, D_i]$ as the estimation of the delivery time given X_i and D_i.
- τ as the treatment effect.

Before applying different models, a visual inspection to the data gives an overview of the trend and sometimes implies which model(s) are appropriate to use. In this study, there are four models of RDD to be discussed as shown in Figure 4.4. Take the upper left figure as an example, which describes an RDD estimation to the dependent variable by a linear model with a common slope. The model could be formulated as

$$\mathbb{E}[Y_i(0)|X_i, D_i] = \alpha + \beta c + \tau D_i + \beta X_i \tag{1}$$

where α represents the intercept of a fitted regression line and β represents the slope. The linear model with different slopes and the non-linear model have similar forms but more parameters. The models are fitted by `lm` function in *R*. The bottom right model shown in Figure 4.4 uses a distinct way from the former three. The model is generated by using `RDestimate` function in `rdd` package of *R*. The function performs local linear regression using the Imbens-Kalyanaraman optimal bandwidth calculation to better fit the data [5]. Instead of fitting the whole bucket of data before or after the intervention, as the former linear or non-linear model does, this function highly focuses in the local area around the cutoff.



(a) Linear model (common slope) **(b)** Linear model (diff slopes)

(c) Non-linear model **(d)** Local linear regression

Figure 4.4: RDD estimation models

[5] `https://www.rdocumentation.org/packages/rdd/versions/0.57/topics/RDestimate`

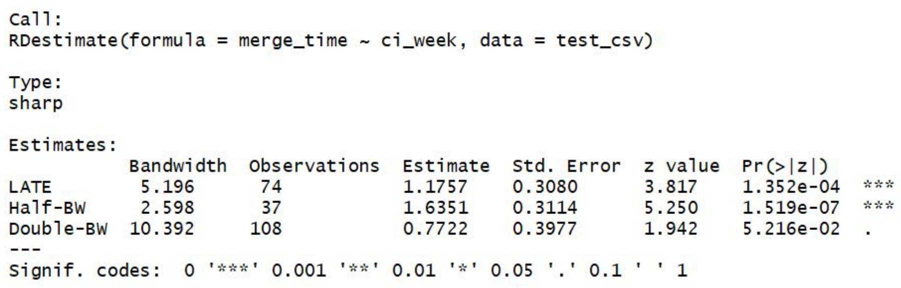
4.2.2.4 Explanation of Sample Results

Except for graphical results like Figure 4.4, the RDD method also generates a statistical report that determines whether the results are significant or not. Table 4.1 gives a summary of a sample linear model with common slope, with the dependent variable of merge time. τ in formula 1 is the value of *ci* in this table, which means that the intervention brings an increase of merge time in 1.6 days with statistically significance at the 5% level, and a standard deviation value of 0.781. The R^2 value also decides the effectiveness of a model because that the higher a R^2 is, the better-observed outcomes are accurate compared with the real values. The summary table of a linear model with different slopes or a non-linear model is similar to Table 4.1. Although such a table includes more parameters, the value of *ci* (the τ) in the formula, the significance level as well as R^2 are the emphases.

Figure 4.5 shows the summary of a model generated by `RDestimate` function in *R*. *LATE* in the table is the abbreviation of the local average treatment effect. The estimation of *LATE* is calculated by the optimal bandwidth, suggests that with statistical significance at the 0% level, the adoption of CI results in 1.1757 days increase in merge time of a PR.

Table 4.1: Sample linear model with common slope

	Dependent variable:
	merge_time
ci	1.600^{**}
	(0.781)
ci_week	-0.004
	(0.008)
Constant	1.936^{***}
	(0.484)
Observations	93
R^2	0.143
Adjusted R^2	0.124
Residual Std. Error	1.500 (df = 90)
F Statistic	7.514^{***} (df = 2; 90)
Note:	*p<0.1; **p<0.05; ***p<0.01



```
Call:
RDestimate(formula = merge_time ~ ci_week, data = test_csv)

Type:
sharp

Estimates:
           Bandwidth  Observations  Estimate  Std. Error  z value  Pr(>|z|)
LATE        5.196      74           1.1757    0.3080      3.817    1.352e-04  ***
Half-BW     2.598      37           1.6351    0.3114      5.250    1.519e-07  ***
Double-BW  10.392     108           0.7722    0.3977      1.942    5.216e-02  .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 4.5: Sample results of `RDestimate` in *R*

4.2.3 Regression Model

Regression Modeling with Ordinary Least Squares is used to replicate and extend the original RQ3, which is the goal of RQ2 of this study. The research question aims to tell factors in the Pull-based development that could partially decide the delivery delay of a PR. Multiple linear regression models are fitted to describe the relationship between explanatory variable X, e.g., code churn, merge time of a PR record, and the dependent variable Y (delivery delay of merged PRs). For each project, I build two models, one that uses the PR data before CI, and another that uses the data after applying CI.

I follow the steps for fitting linear models from Harrell Jr.'s book[35]. Figure 4.6 shows an overview of the workflow.

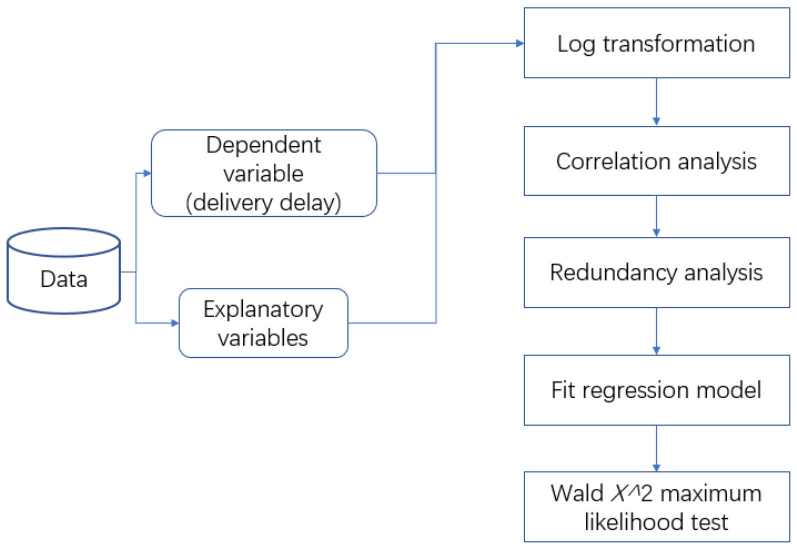


Figure 4.6: The process of fitting a linear regression model.

Step1: transform the response variable (delivery delay) by log function. In metadata, the scale of delivery delay of PRs is enormous, from 0 to thousands in days and the skewness of all delivery-delays is a high value of 27. Log transformation is a common and simple way of pre-processing data in order to reduce skewness [36].

Step2: do a similarity check between every pair of the explanatory variables. The function `varclus` of `Hmisc` package in *R* implements Spearman correlation test and supports hierarchical variable clustering analysis. For variables within a cluster that have a correlation of $\rho^2 > 0.7$, the variable with least skewness within a cluster is remained.

Step3: conduct redundancy check. Spearman's test in the previous step can detect monotonic but nonlinear relationships. A redundancy check is able to detect non-linear relationships that a variable can be easily predicted from all other predictors by using flexible parametric additive regression models [35]. I use the default value of 0.9 as the cutoff of R^2. Redundant variables are removed for later analysis.

Step4: fit regression models. The R^2 is used to assess the fit of a model because it statistically measures how close the data are to the fitted regression line. It explains the percentage of the response variable variation that is explained by the model. Similar to the original study, only regression models that obtain R^2 values higher than 0.5 are seen as valid. From 27 models in total, 18 use data of PRs without CI adoption, and the other 9 use data of PRs with CI applied are valid. `ols` function in *R* is applied.

Step5: use the Wald X^2 maximum likelihood test to get explanatory power of each variable of the 27 valid models. The larger X^2 value of a variable, the larger influence the variable has. The X^2 is calculated by `anov` function of package `rms` in *R*.

5

Results

This chapter presents the results of each research question (RQ). Since the study subject is almost the same as the original study, not only does every section illustrate the results of its own, but a comparison to the original outcomes.

5.1 RQ0 Results

To gain a deeper and detailed understanding of the study of Bernardo *et al.*, a replication study is conducted.

5.1.1 RQ1 in the Original Study

The RQ1 of the original study investigated the impact of delivery delay of PRs by adopting CI. They analyzed three metrics, which are *delivery delay* (days between when a PR got merged and when it was released), *merge delay* (days between when a PR was submitted and when it was merged), and *lifetime* (delivery delay+merge delay). As stated in Section 4.2.1, the authors applied two non-parametric tests to analyze the distribution of the data, and so does this study. As expected, the replication results are very close to the original results, and the differences are shown in Table 5.1. For metric *merge delay* and *delivery delay*, the results of the proportion of projects observed significant changes, the median Cliff's delta value, and the proportion of projects which had a smaller value of the metric are listed sequentially. The median Cliff's delta of metric *lifetime* was however missed in the original study.

Note that even the same metadata is fed into the same statistical methods, the results are not exactly the same. The reason could be the authors may apply different data pre-processing procedures that are not described in the paper. Both the original results and replication outcome support the claim that after the CI adoption, in most projects the merge time of a PR got faster, yet either the delivery delay or the lifetime did not show significant changes which contradict the assumption that the introduction of CI should fasten the delivery process.

Metric	Original Result	Replication Result	Diff(abs)
Delivery delay	82.8%	83.9%	1.1%
	0.304	0.284	0.02
	51.4%	47.9%	3.5%
Merge delay	72.4%	78.2%	5.8%
	0.206	0.195	0.011
	27%	29.4%	2.4%
Lifetime	71.3%	72.4%	1.1%
	48.4%	52.4%	4%

Table 5.1: Comparison between the original results and the replication results.

5.1.2 RQ2 in the Original Study

Following RQ1, RQ2 in the original study then tried to find the reason for this phenomenon by observing the development activities per release of a project before and after the CI adoption. The authors applied the Mann-Whitney-Wilcoxon test and Cliff's delta again to measure all the metrics, along with using a box plot to visually summarize the comparison. Firstly, they analyzed PR submission, merge, and delivery rates per release. Figure 5.1 is from the original paper, showing 3 boxplots for the metrics respectively. Figure 5.2 on the right is from this study. The number shown in each box is the median value of the distribution, and the difference of each pair of median values from the original study and replication study is 0 for every metric. The results tell that after CI was adopted, the number of submitted, merged and released PRs per release increases. Similar to Table 5.1, Table 5.2 shows the difference of the statistical tests. The first row of each metric gives *p-value* of the observed increases in metrics after CI, and the second row lists the value of Cliff's Delta. Since the distribution differences of metric *releases/ year* are not significant (*p-value* = 0.146 / 0.301), Cliff's Delta was not computed.

Although all the *p-values* and Cliff's deltas are not exactly equal with the original results, they show similarity that both the values shows strong significance or a medium magnitude of the difference between the two distributions. All the significance values are far less than 0.05 which means the increase of each metric is always significant. Then they analyzed the sum of code churn per release of each project. The code churn is the sum of added and deleted lines of code of all commits in one PR. The authors also observed a significant increase in this metric, so does the replication study.

However, they did not observe significant changes in released frequency after CI. Figure 5.3 shows that from the two research results, both two data distributions are of small difference, and the p-values presented in Table 5.2 gives statistical evidence of this.

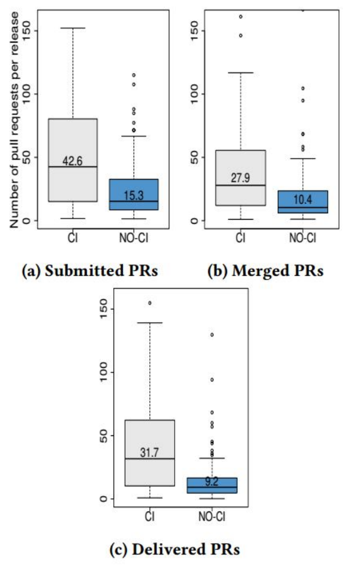


Figure 5.1: Results from the original study.

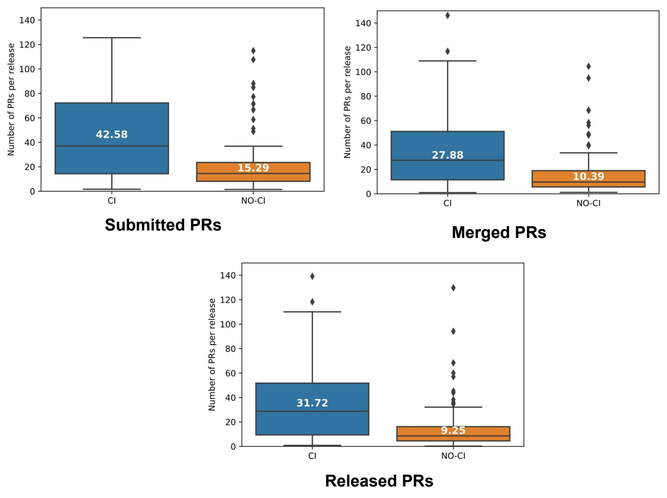
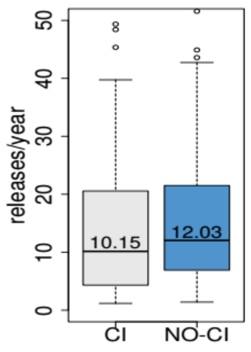


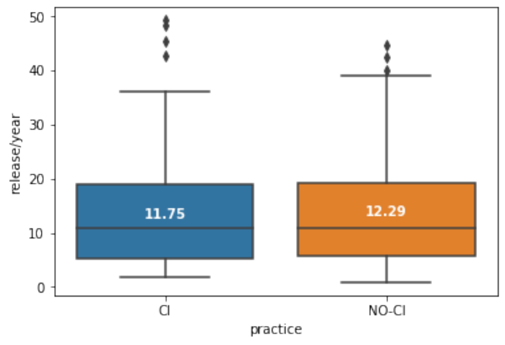
Figure 5.2: Results from the replication study.

Metric	Original Result	Replication Result	Diff(abs)
PR submission/ release	1.5e-04	7.73e-05	7.27e-05
	0.332	0.332	0.0
Merged PR/ release	7.9e-05	3.9e-05	4e-05
	0.347	0.347	0.0
Delivered PR/ release	1.37e-05	6.83e-06	6.87e-06
	0.382	0.382	0.0
Churn/ release	2.27e-03	1.1e-3	1.17e-3
	0.27	0.27	0.0
Releases/ year	0.146	0.301	0.155

Table 5.2: Comparison between the original results and the replication results.



(a) Original result



(b) Replication result

Figure 5.3: Release/year before and after CI

The replication study successfully reproduces the results in the original study with acceptable minor deviations. This task gives me a full understanding of the motivations, approaches, and outcomes of the original study, thus lays a solid foundation for continuing researching on the following RQs.

5.2 RQ1 Results

Owing to the concern of hidden patterns in data may impact the original results, the first RQ in this study is proposed to check whether similar results of RQ1 and RQ2 can be found when applying an alternative statistic method to the projects and dataset in the original study. The Regression Discontinuity Design (RDD) method is applied to investigate whether the intervention of CI adoption influences the metric performances on the scale of time.

For the RQ1 of the original study, there are three metrics: delivery delay, merge delay and lifetime. To choose a suitable model for RDD analysis, firstly I generate scatter plots and conduct a visual inspection for every project. Figure 5.4 contains sample results from 3 different projects, **mantl/mantl**, **aframevr/aframe**, and **mozilla-b2g/gaia**. The x-axis represents the forcing variable, which is the number of weeks between the CI adoption time and the created time of a merged PR record. The y-axis is the delivery delay of a PR in days. There is an obvious pattern that the scatter points compose many line segments with a negative slope. This pattern exists in these three projects, and almost all the 87 projects. The explanation of the phenomenon is that delivery time of a merged PR is heavily dependent on when the integrators decided to launch a release. A straight conclusion from the visualization result is that there seems no universal pattern regarding time in the metric delivery time. Similarly, the lifetime metric has the pattern also, as shown in Figure 5.5. Therefore, RDD is not applicable to the two metrics.

But such a pattern does not exist in the merge time metric by visual inspection. And it's hard to tell if a linear model or a non-linear model can fit the data better simply from the scatter plots. Hence, all four models described in Section 4.2.2 are applied. The data of a project is divided into two buckets by the cutoff point of when CI was adopted, and one model fits all data in one bucket is built. Figure 5.6 shows the fitted models of project **boto/boto**. In the first three models, the red and blue lines fit data after and before the intervention respectively.

Among 87 projects, none of the linear or non-linear models gives a value of R^2 higher than 0.35. It means a very low proportion of the variance for the merge time that could be explained either by the time difference from the time introducing CI, or whether a record applying CI, or the interaction between the former two variables. So, the conclusion that there is no pattern regarding time exists for merge time metric is drawn. Regardless of the small R^2, 63% (55/87) of the projects suggest that PRs before CI adoption merge faster, which is consistent with the original study. As

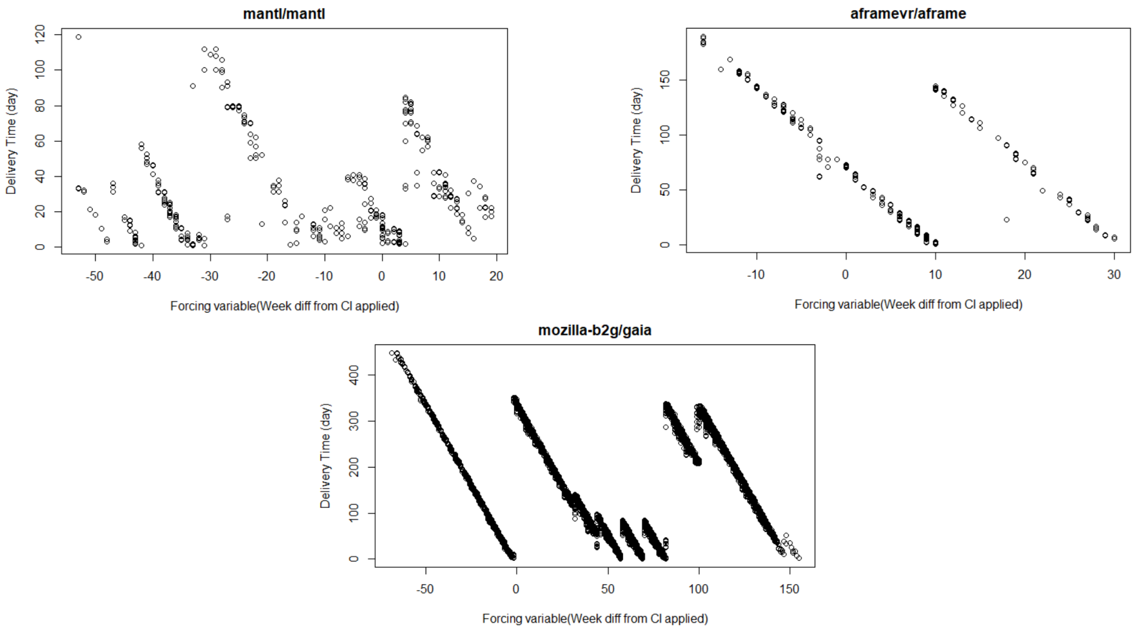


Figure 5.4: Visual inspection of metric *delivery delay*

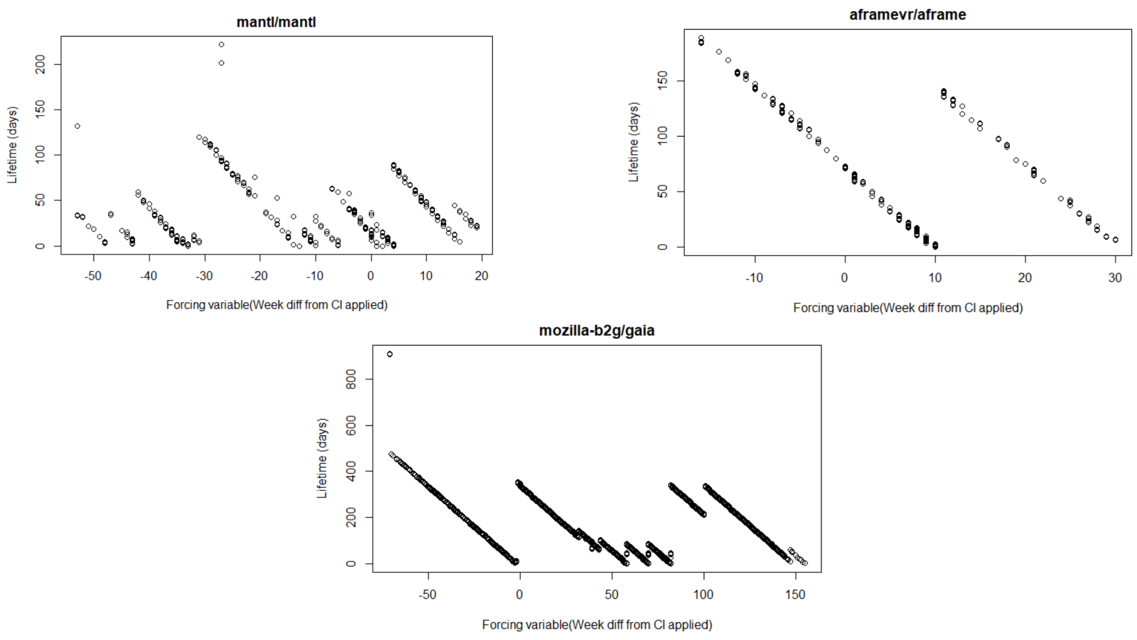
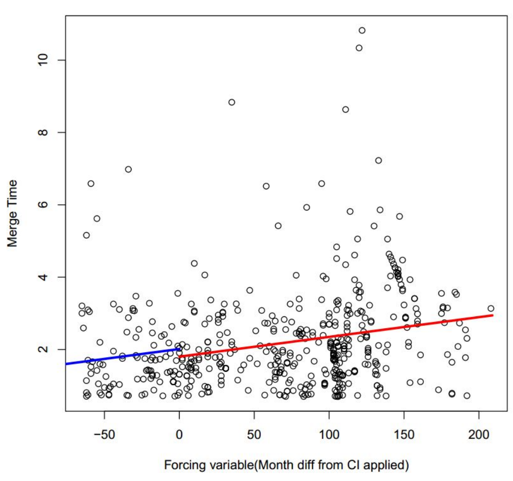
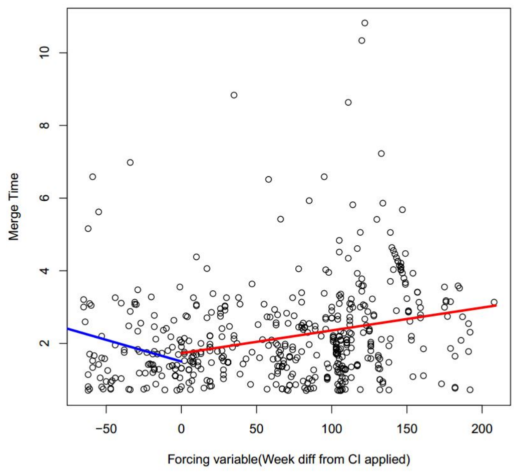


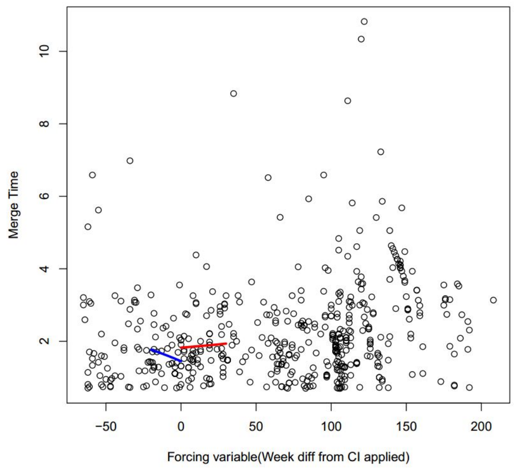
Figure 5.5: Visual inspection of metric *lifetime*



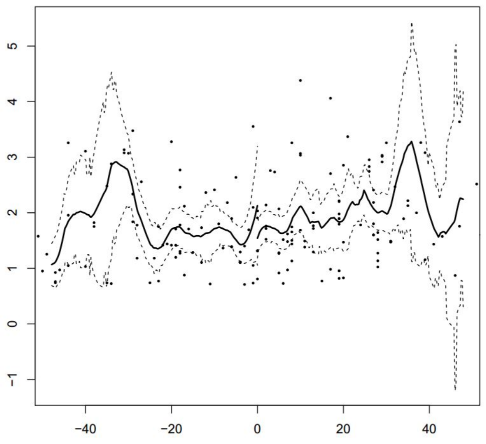
(a) Linear model (common slope)



(b) Linear model (diff slopes)



(c) Non-linear model



(d) Local linear regression model

Figure 5.6: RDD models of project **boto/boto**

for the results of `RDestimate` function, 65/87 valid RD models are got–the reason for 22 of them not being valid is that internal errors occur when estimating bandwidth in the calculation. 47.7% (31/65) projects show significance of the estimation at the 5% level (*p-value* = 0.05). 61.3% (19/31) of models give a positive estimation of *LATE* (local average treatment effect) around the cutoff which means that PRs merge slower after CI adoption.

The metrics of RQ2 in the original study are *submitted, merged, delivered number of PRs per release, code churn per release* and *number of releases per year*. The number of data records equals to the number of releases of each project, thus the data size is too small to apply RDD. Alternatively, I check the release frequency before and after the CI adoption by a different metric which is the number of days between two consecutive releases. As Figure 5.7 shows, the median of days between two releases before CI adoption is 35.23 with a median value of 33.75 after applying CI. The MWW test gives a p-value of 0.115 > 0.05, which means the difference between the two distributions is not significant. The result is consistent with the original study.

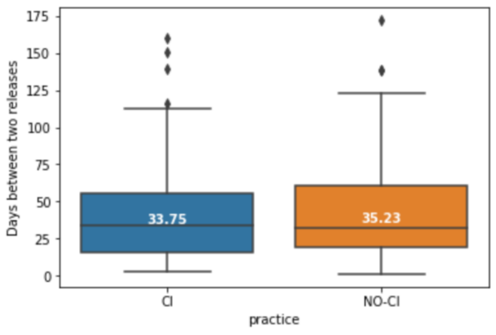


Figure 5.7: Box plot of days between two releases

5.3 RQ2 Results

The RQ2 in this study explores new metrics that have strong explanatory power if it is added to the linear regression model whose dependent variable is *delivery delay* of PRs. In the original study, following the guideline of fitting model in Harrel Jr.'s book [35], the authors built two models for each project. One model uses PR data before CI, another uses PR data after CI. The authors used 13 metrics in every model, 6 of them showed a certain extent of explanatory power to the variation to *delivery delay* according to Figure 10 and Figure 11 in the paper [8]. Based on their results, variables without obvious influence on the models are removed from this analysis work. The remaining variables from original study are *merge workload*, *contributor experience*, *queue rank*, *contributor integration*, *activities*, and *merge time*. Table 5.3 lists the detailed definition and rationale of each variable. In this study, a new metric named *come in time* is introduced. As Figure 5.8 shows, it is the time interval in days between the time when a PR got merged with the begin of the release cycle in which the PR was delivered.

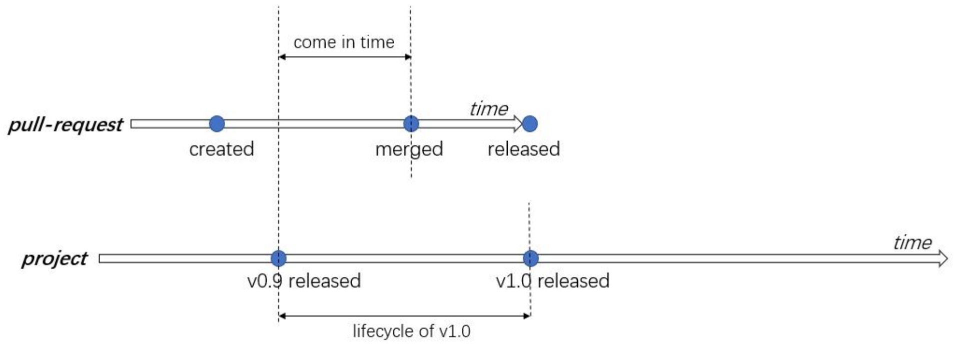


Figure 5.8: The definition of metric *come in time*

By following the steps described in Section 4.2.3, I get 17 out of 86 valid models that are fitted form PR data without CI and have R^2 values higher than 0.5. The median of the R^2 values is 0.58. 9 out of 86 are valid models using the data of PRs with CI adoption. The median of R^2 values is 0.57. These two median values are both lower than the results of the original study, and a possible reason could be that the more predict variables, the higher a R^2 gets. There were 13 predictors in the original study, while in this study I have 7 factors, that 6 of them were proved very influential, 1 of them is newly introduced. Figure 5.9 shows the number of projects for one explanatory variable that has the strongest influence. Take the metric *merge workload* for example, it shows the strongest influential power in 3 models that are fitted from PR data before CI. Figure 5.9 tells that either for PR data before CI or after CI, the metric *come in time* is the most influential attribute to the variation of *delivery delay*.

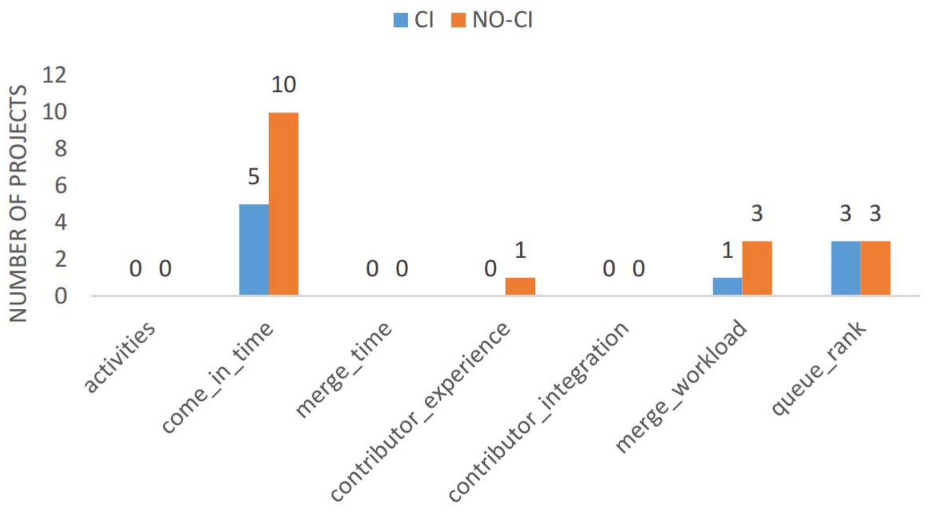


Figure 5.9: The number of models for the explanatory variables that shows the strongest influence

Factor	Definition (**d**) | Exposition (**e**)
Number of Activities	**d**: An activity is an action to a PR conducted by a GitHub user, eg. labeled, assigned.
	e: Generally a high number of activities represents plenty of discussion rounds between the integrator and contributors [37].
Merge Time	**d**: The time interval between when a PR was created and when it was merged by an integrator.
	e: If a PR was merged quickly, it is also likely to be released quickly, maybe because that the PR fixes a serious bug.
Contributor Experience	**d**: The number of released PR that were created by author of a certain PR.
	e: If a user has previously contributed a lot released PRs to the project, the next PR submitted by the person would be seen very valuable and thus got merged and released quickly [8].
Contributor Integration	**d**: The mean value in days of the released PRs that were submitted by a particular user.
	e: If the past PRs submitted by a particular contributor were released quickly, then the next PR submitted by the same user would also be released rapidly [8].
Merge Workload	**d**: The number of PRs waiting to be merged at the time when the certain PR was submitted [8].
	e: Since the energy of an integrator is limited and each submitted PR requires analyzed, the more unmerged PRs, the heavier workload of an integrator would have. This might impact the delivery of PRs. [8]
Queue Rank	**d**: The order of the merged PRs in a release cycle. For example, the first merged PR in the release cycle is assigned with a value of 1, the hundredth one is assigned with 100 [8].
	e: A merged PR might be released faster or slower depending of its queue position [38].
Come in Time	**d**: The time interval in days between the time when a PR got merged with the begin of the release cycle in which the PR was delivered.
	e: The earlier a PR comes in the release cycle, the longer it would take for the PR to be delivered.

Table 5.3: Definition and Exposition of each Variable in Linear Regression Model.

5.4 RQ3 Results

In the original study, though the delivery time of merged PRs was not impacted by the introduction of CI, increasing trends of other metrics including merge time of submitted PRs, the number of submitted, merged PRs per release, *etc.* was observed with statistical significance after CI. The study of this RQ aims to control over the projects that never apply CI in their lifetime and inspects the results. If similar results are generated from this study, then the adoption of CI is not a unique factor that impacts the development activities in open source projects. As Section 4.1.1 states, a *mocked-CI-point* referring to the time when 38% of a project's lifetime was spent is introduced, functioning as an intervention. *mocked-CI* is distinguished from CI for clarity in this Section. 28 valid projects from GitHub are collected and analyzed using all methods applied in former RQs.

5.4.1 Changes in Time Spans of PRs after *mocked-CI-point*

By applying Wilcoxon signed rank test, among 28 projects, 25 (89.3%) of them obtain significant *p-value* when comparing the merge time of submitted PRs before and after mocked-CI with a median Cliff's delta value of 0.266 (small). The majority projects (18 out of 25) merge PRs more quickly before mocked-CI. For delivery time and lifetime of PR, I do not observe a change of the data distribution in the majority. A total of 96.4% (27/28) of the projects gain significance on the time of a merged PR waiting to be delivered along with a median of delta value of 0.292 (small). Only 59.2% of the projects deliver PRs faster after mocked-CI. Considering the small number of analyzed projects, a phenomenon in the 59% could not represent the majority. In 25 (89.3%) out of 28 projects significant *p-values* as well as a median of Cliff's delta of 0.35 are observed. 60% (15/25) of them even have longer lifetime after mocked-CI. All differences in lifetime distribution of the 25 projects are non-negligible according to Cliff's delta results. 44% (11/25) of such projects obtain small delta values with a median of 0.249, 16% (4/25) are observed having medium deltas with a median of 0.357, and the rest 40% (10/25) of them acquire a median of 0.651 among 10 large delta values.

Following the RQ1 of this study, the alternative method RDD is again used in this task to detect time series patterns in metric *merge time*. A linear model with common slope, a linear model with different slopes, a quadratic model and a model generated by `RDestimate` function in *R*. The validity of linear and non-linear models is evaluated by R^2 value. However, for each project none of the models obtain a R^2 value higher than 0.4, meaning that more than 40% of the variation in merge time is not explained by the model. All models are under-fitted, indicating that the relationship between merge time with the time series factor regarding CI is neither linear nor quadratic. Thus there is no time series pattern existed in the metric merge time. Regardless of R^2 values, 57.1% (16/28) of the models suggest

PRs after mocked-CI merge faster. The `RDestimate` function generates models for 17 projects. Only 29.4% (5/17) of the models give significant estimation at the 5% level to the changes after mocked-CI. Put the significance aside, 64.7% (11/17) imply after mocked-CI, the submitted PRs get merged faster.

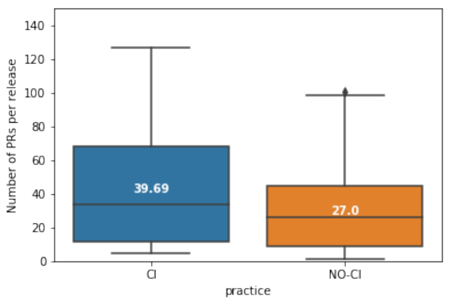
The non-parametric results in the 28 projects without CI are consistent with the findings in the original paper regarding these three metrics, that the merge time and lifetime of submitted PRs is even longer after the time node, and the acceleration in delivery time of merged PRs is significant only in approximately half of the total projects. However the outcome of RDD gives different results from the original 86 projects and the projects without CI that the merge time of PRs significantly increase after CI in a majority of projects, whilst a decline is observed in most 28 projects without CI after the mocked-CI.

5.4.2 Changes in Development Activities after *mocked-CI-point*

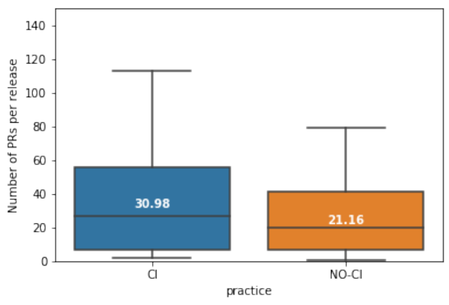
The first 3 boxplots of Figure 5.10 show the number of created, merged and released PRs per release before and after mocked-CI respectively. Two metrics show increases of the median values after mocked-CI, that the median value of submitted PRs increases from 27.0 to 39.7 and the median value of merged PRs increases from 21.2 to 31.0. For delivered PRs per release, although the upper quartile, which is the upper bound of a box meaning 75% of data fall below the line, increases after mocked-CI, the value of median drops by 6.95. From a statistical perspective, none of the metrics shows significance. A Wilcoxon signed rank test reveals that the increase in the number of submitted PRs is not significant (*p-value* = 0.09085 > 0.05), with a small Cliff's delta of 0.209. The number of merged PRs gets a bigger *p-value* of 0.121 by the MWW test, with a small delta of 0.184. A non-significant value (*p-value* = 0.294) is also observed in the MWW test for the number of released PRs per release, as well as a negligible Cliff's delta value (0.085). The sum of churns in submitted PRs per release almost doubles (from 13,229 to 23,428) when observing the median value before and after mocked-CI. The *p-value* = 0.033 < 0.05 suggests the increase is significant, with a small Cliff's delta of 0.288.

Comparing to results in the original study (check Figure 5.1) that the observed increases in submitted and merged PRs per release are statistically significant, but the increases observed from projects with mocked-CI are not significant. The discrepancy is bigger when comparing the released PRs per release, that in original study the number of released PRs per release increased significantly (*p-value* = 2.27e-03) after CI, while the difference in the distribution before and after mocked-CI observed here is not significant. For the sum of PR code churn per release, both studies observe a significant increase after the selected time node.

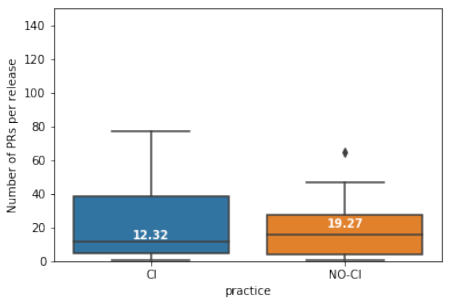
Figure 5.10d shows the distribution of the number of releases per year before and



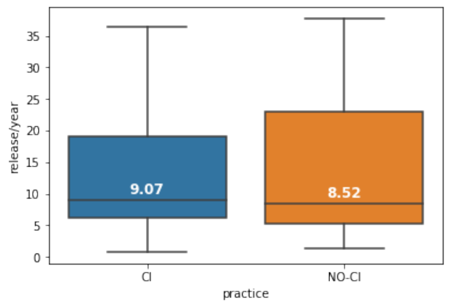
(a) #Submitted PRs/release



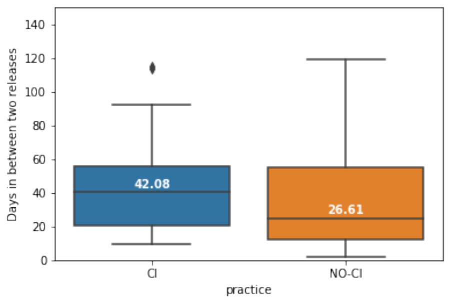
(b) #Merged PRs/release



(c) #Delivered PRs/release



(d) #release/year



(e) #Days between two releases

Figure 5.10: Results of RQ3

after mocked-CI. The projects tend to launch 9.07 releases per year after mocked-CI and a slightly smaller number of 8.52 before mocked-CI in the median. A Wilcoxon signed rank test shows that the increase is far from significant with *p-value* = 0.286. Similarly, the result in the original study also gives an insignificant *p-value* of 0.146 with a drop from 12.03 to 10.15 number of released shipped per year. Another metric that evaluates the release frequency computes the time difference in days between two consecutive releases is introduced in Section 5.2. As Figure 5.10e shows, the time interval between releases increases from 26.61 to 42.08 in the median. Since fewer releases are shipped per year, the time interval between two releases increases. The phenomenon is not significant with a *p-value* of 0.138, which is consistent with the metric *release per year*.

5.4.3 Most Influential Variables in Linear Regression Models

Following Section 5.3, the task of this section is fitting linear regression models to find out which factors can impact the delivery time of projects that never apply CI. The analyzed variables are *activities*, *come in time*, *merge time*, *contributor integration*, *contributor experience*, *merge workload*, and *queue rank* as detailed explained in Table 5.3. The analysis procedure follows the approaches defined in Section 4.2.3, that filtering redundant variables goes first, then fitting a linear regression model, finally getting the explanatory power of each variable in the model by a Wald X^2 maximum likelihood test. In 28 models that use data after mocked-CI, only 2 models have R^2 value higher than 0.5. One of them obtains a R^2 of 0.929 with *queue rank* as the most influential variable, another obtains a R^2 of 0.528 with *contributor experience* as the most powerful one. In models using data before mocked-CI, 3 of them have R^2 value higher than 0.5 (0.729, 0.592, 0.810), with *queue rank* being the most influential variable. *queue rank* always shows the strongest explanatory power both in models that fit data before or after mocked-CI. It represents the sequential order of merged PRs in a release cycle (see Table 5.3). And *contributor experience* infers the number of successfully delivered PRs submitted by a particular contributor previously. These two factors are of the biggest explanatory power to the variation in delivery time of PRs of projects that never apply CI.

5.4.4 Comparing to the Original Results

Considering the performance of the projects that never apply CI from all the metrics, the divergence between them and the previous investigated 87 projects is not trivial. In merge time of PRs, the authors claimed that 73% of the projects tended to merge PRs slower after CI and results of RDD from this study Section 5.2 seconds them. However in 28 projects without CI applied, the merge time of PRs after mocked-CI drops according to the analysis of RDD. For the release related development activities, the authors observed a significant increase in the number of submitted, merged and delivered PRs after adopting CI. In this study, increases are also observed, yet non-significant. Lastly, the original study found that *merge workload* and *queue rank* are the most influential variables in linear regression models using data before and after CI respectively. However, the variable that has the biggest explanatory power is *queue rank* either for models analyzing data before or after the mocked-CI time node. Therefore from the comparison, it is recognized that the adoption of CI makes the projects distinct from the projects that do not apply CI.

5.5 RQ4 Results

This RQ aims to test the generalizability of findings in the original paper by collecting 54 new projects adopted CI from GitHub then conducting analysis. Similar outcomes as the original results are expected naturally.

5.5.1 Changes in Time Spans of PRs after CI

Firstly applying Wilcoxon signed rank test to the distributions before and after CI and measuring Cliff's delta to the difference magnitude as the original study did. The statistical results are listed in Table 5.4. The table also lists outcome from the original paper as a reference, for showing the differences between two result-sets. The results of the proportion of projects observe significant changes, the median of Cliff's delta value, and the proportion of projects which have a smaller value of the metric after CI are listed sequentially. The median Cliff's delta of metric *lifetime* was however missed in the original study. In general, the discrepancy of one metric for the two datasets containing different projects are small, thus one can claim that the validity of the original finding holds in this study.

Metric	Original Result	RQ4 Result
Delivery delay	82.8%(72/87)	98.0% (50/51)
	0.304(small)	0.254(small)
	51.3%(37/72)	58%(29/50)
Merge delay	72.4%(63/87)	80.4%(41/51)
	0.206(negligible)	0.188(negligible)
	27%(46/63)	19.5%(33/41)
Lifetime	71.3%(62/87)	79.6%(43/54)
	48.4%(30/62)	46.5%(20/43)

Table 5.4: Comparison between the original results and RQ4 results.

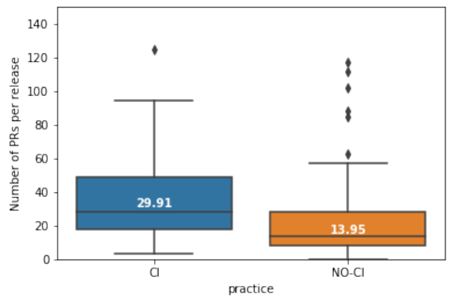
Then RDD is applied to inspect time series patterns in merge time with four models (two linear models, one non-linear model and one generated from `RDestimate` function). The R^2 values of all linear, nonlinear models are smaller than 0.4, indicating that all models underfit the data thus there is no time series pattern existing in metric *merge time*. Regardless of R^2, that 57.4%(31/54) of the projects merge PRs faster after CI is observed. `RDestimate` function successfully generates models for 41 projects, and 56.1% (23/41) of them outputs estimation at the 5% level of significance (*p-value* = 0.05). A total of 60.9% (14/23) of the models give negative estimation meaning that the time for a submitted PR waiting to be merged gets smaller after CI.

By conducting non-parametric tests as the original study did, findings from the 54 projects with CI are still consistent with the outcome of the paper. In other words, the original results are proved valid regarding metrics of merge time, delivery time and lifetime of PRs. But analyzing the new dataset by RDD reveals an opposite consequence that instead of an increase in merge time of PRs after CI, a decrease is observed in over half of the projects.

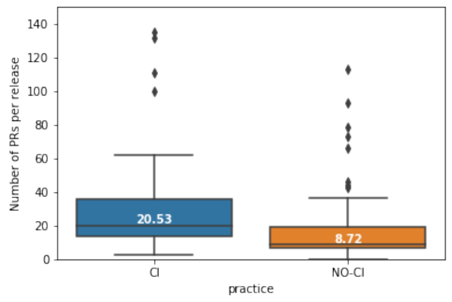
5.5.2 Changes in Development Activities after CI

In Figure 5.11, the first 3 plots show the number of submitted, merged and released PRs per release before and after CI introduction. Each of the plots gives an overview of an increase in the data that the location of the "box" moves upward on y-axis after CI. In further detail, the median value of submitted PRs increases from 13.95 before CI to 29.91, the median of merged PRs per releases also experiences an increase from 8.72 to 20.53, and the number of released PRs increases to 16.1 from 7.59. All increases are significant, which are supported by the MWW rank test with *p-values* far less than the threshold 0.05 and Cliff's delta values in between 0.33 and 0.47 of medium magnitude. Besides, the median value of the sum of code churns in submitted PRs within a release cycle after CI (15,715) doubles the value before CI (7,453). The change is significant supported by a *p-value* = 6.7e-03 with a small Cliff's delta (0.276). Table 5.5 shows the comparison between the results from the original paper and from this study. The table lists *p-value* from MWW test in the first rown and the median of Cliff's delta in the second row for each metric. Regarding the five metrics, one can tell the two result-sets have small gaps.

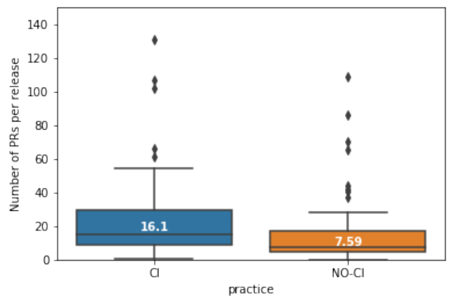
The boxplots in Figure 5.11d shows the distribution of the number of releases per year before and after CI. The projects tend to deliver 7.97 releases per year after CI compared to a slightly bigger value of 8.98 before CI in the median. An MWW test gives a *p-value* of 0.409 showing non-significance in the changes of the two distributions. Similarly, the original paper also claimed that there was a small drop from 12.03 to 10.15 in the median of the number of shipped releases per year with an insignificant *p-value* = 0.146. Figure 5.11e shows an increase from a median of 35.9 to 45.7 of the time interval in days between two continuous releases after CI. Still, the increase is not significant (*p-value* = 0.123) with a negligible Cliff's delta.



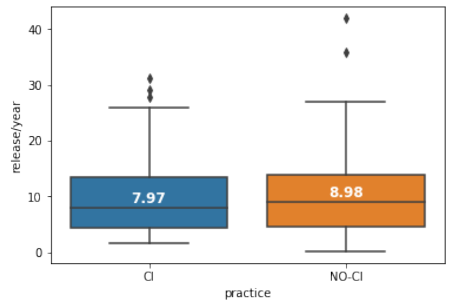
(a) #Submitted PRs/release



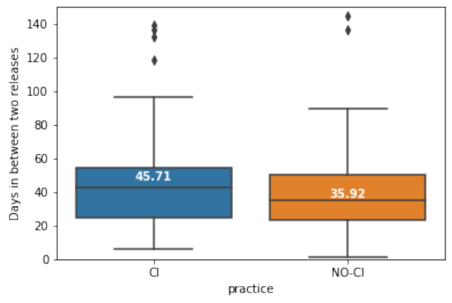
(b) #Merged PRs/release



(c) #Delivered PRs/release



(d) #Release/year



(e) #Days between two releases

Figure 5.11: Results of RQ4

Metric	Original Result	RQ4 Result	Diff(abs)
PR submission/ release	1.5e-04	3.67e-04	2.17e-04
	0.332	0.377	0.055
Merged PR/ release	7.9e-05	2.1e-04	1.31e-04
	0.347	0.394	0.047
Delivered PR/ release	1.37e-05	8.19e-04	8.05e-04
	0.382	0.352	0.03
Churn sum/ release	2.27e-03	6.7e-03	4.43e-03
	0.27	0.276	0.006
Releases/ year	0.146	0.409	0.263

Table 5.5: Comparison between the original results and RQ4 results.

5.5.3 Most Influential Variables in Linear Regression Models

In this section, I analyze the explanatory power of each variable, then compare the results to the original findings. Similar to Section 5.3, I computed 7 metrics and fitted linear regression models with the dependent variable as the delivery time. In 54 models that use data after CI, 2 models are regarded valid as their R^2 values are bigger than 0.5. One of them obtains a R^2 of 0.696 with *contributor integration* as the most influential variable, another gets a R^2 of 0.537 with *queue rank* being the most powerful one. In models using data before CI, only 5 of them obtain R^2 values higher than 0.5 (0.985, 0.552, 0.904, 0.561, 0.513). 3 out of 5 models result in *queue rank* as the most significant factor for impacting the delivery time, the other models reveal that *contributor integration* and *merge workload* are also able to alter delivery time.

If relaxing the threshold of R^2 to 0.3, meaning that at least 30% of the variability of my data is explained by a linear regression model, more models are considered valid. In models using data after the adoption of CI, 16 are regarded as valid with a median R^2 of 0.349. On the other hand, 19 out of 54 models using data before CI are valid with a median R^2 of 0.415. Table 5.6 shows the number of models per the most influential variables for data before or after CI. Still, *queue rank* and *contributor integration* are the first and second most powerful factor either for models fitting the data before CI or after CI. *contributor experience* and *merge workload* also get their time of being the most powerful variables.

practice	queue rank	contributor integration	contributor experience	merge time	merge workload
CI	7	6	1	0	2
NO-CI	8	7	1	1	2

Table 5.6: The number of projects per the most influential variables

5.5.4 Comparing to the Original Results

Most of the statistical results from the new 54 projects second findings from the original paper. By non-parametric tests, half (58%) of the new projects deliver merged PRs faster after CI, comparing to a value of 51.3% in the old projects. The increases in the number of merged delivered and submitted PRs are significant as found in the original study. Although there is a divergence in which one influences the delivery time of PRs by fitting linear regression models using data before CI, that in the original study, it is *merge workload*, yet in this study, the answer is *queue rank*, followed by *merge workload* which obtains the second strongest explanatory power. Therefore, the new projects with CI are observed with similar statistical performance as the old projects.

6

Threats to Validity

This chapter addresses the potential threats to validity in this thesis work.

6.1 Construct Validity

Construct validity would be a threat if the research design has flaws or inaccurate so that it is not able to measure what it claims or purports [39]. The statistical methods including two non-parametric tests (MWW test and Cliff's delta), linear regression modeling and Regression Discontinuity Design (RDD) are considered old and mature methods. The implementations of the methods are either supported in R or in Python, and one can believe that the implementations have been tested and verified by the software provider. Hence a construct threat to validity might be introduced by errors in the processes of collecting projects and metadata. Before collecting new dataset, I have tried to collect raw data and compute metrics for one of the 87 projects used in the original study then compared my results to the metadata that the authors publish online, for the correctness and a full understanding of their approaches. Also from time to time, I verify the intermediate results during the process of collecting.

6.2 External Validity

External validity concerns to what extent do the findings still hold under a more generalized circumstance [39]. In this study, 52 new projects with CI adopted have been analyzed to test the generalizability of the results in the original paper, and 30 projects without CI are investigated for the purpose of controlling the impacts of CI. But the amount of investigated objects is small compared to the number of available data online. Besides, it could be interesting that applying the methodologies in this paper to datasets with different settings, for example, a dataset covering private projects.

6.3 Internal Validity

Internal validity questions the ability to draw conclusions from all the involved factors, that if any of the factors are potentially affected by other unexpected variables. One threat could be that the purpose of each collected project may influence the release activities, yet is not analyzed in this study. Commercial software releases regularly and fast to end-users, e.g. Amazon delivered to production every 11.6 seconds on average in May 2011, Facebook released twice a day [40]. Non-commercial projects, which are maintained by core contributors from enthusiasm, may release less regularly. Should differentiate projects according to its purpose, the influence brought by CI could be more obvious.

7

Conclusion & Future Work

This thesis work is motivated by the study of Bernardo *et al.* which investigating the influence brought by adopting CI on the delivery delay of PRs empirically. The goal of this study is to replicate the findings, extend the analysis scope and test the validity of the findings under different settings. Here I summarize each RQs of this study.

RQ0: *Can the original results be reproduced?*
Firstly to fully understand the work of Bernardo *et al.*, I have learned the methodologies they applied and replicated the main results. The results are described in Section 5.1. By comparing the results I have obtained to the original results, I can safely claim that I have understood the original study enough which lays a solid foundation for later analysis work.

RQ1: *Can similar results to the original study are found when applying an alternative statistical method?*
RQ1 was proposed by concerning for validity in their study that they did not try investigating hidden pattern rooted in data. In order to dispel the concerns, Regression Discontinuity Design, which analyzes the impact of an intervention (in this case, the adoption of CI) by a time series analysis, is applied for detecting such pattern. The results show that there is no time series pattern existing in the analyzed metrics from the original dataset, and the original results are supported by this method.

RQ2: *Are there any other metrics that have strong explanatory power if added to the linear regression model in the original paper?*
The work of RQ2 tries to figure out any metric other than those in the original study could also influence the delivery time of PRs by fitting linear regression models. I came up with the new metric *come in time*, and it shows overwhelming explanatory power among all other metrics.

RQ3: *Can similar results be found when applying the same methodology to projects that never introduced CI?*
RQ3 was analyzed for testing the credibility of the original results by controlling over projects that never apply CI. I collected 28 new projects from GitHub and computed needed metrics. After conducting all methods that are used in the former

tasks, I observe that the statistical performance from the projects without CI is different from the projects which are investigated in the original study. Therefore the adoption of CI is a unique factor in impacting the delivery time of PRs as well as other development activities.

RQ4: *Can similar results are found when applying the same methodology to different projects that introduced CI in their lifecycle?*
The final RQ aims to test the generalizability of the findings of the original study. I collected 54 projects with CI adopted and analyzed them. The outcome shows that the new projects have similar statistical performance as the old projects, that only half of the projects deliver PRs faster after CI, and the development activities increase significantly.

7.1 Future Work

Either by applying an alternative method (RDD) or expanding the study objects (projects that never apply CI), results in the original study are proved solid and extensible. Still, further research on this topic is necessary. This study only uses data from public open source projects. In the future, if circumstances permit, one can investigate private projects (e.g. private projects from GitHub, private enterprise projects). Besides, further study is needed considering the possible impact of the factor that whether a project is commercial-oriented as stated in Section 6.3. Finally, regarding the study which controls over projects never apply CI, this study chooses a time node of when 38% lifetime has spent based on the mean and median value of when CI was adopted in the 87 old projects. One could try setting another time point in the analysis. For example, Hilton *et al.* claimed that the median time before CI was adopted was around one year [19].

Bibliography

[1] (2000) Continuous integration (original version). [Online]. Available: https://martinfowler.com/articles/originalContinuousIntegration.html

[2] P. M. Duvall, S. Matyas, and A. Glover, *Continuous integration: improving software quality and reducing risk.* Pearson Education, 2007.

[3] M. A. Cusumano, "Extreme programming compared with microsoft-style iterative development," *Communications of the ACM*, vol. 50, no. 10, pp. 15–18, 2007.

[4] J. Holck and N. Jørgensen, "Continuous integration and quality assurance: A case study of two open source projects," *Australasian Journal of Information Systems*, vol. 11, no. 1, 2003.

[5] B. J. Dempsey, D. Weiss, P. Jones, and J. Greenberg, "Who is an open source software developer?" *Communications of the ACM*, vol. 45, no. 2, pp. 67–72, 2002.

[6] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, "Quality and productivity outcomes relating to continuous integration in github," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*, 2015.

[7] D. Ståhl and J. Bosch, "Modeling continuous integration practice differences in industry software development," in *Journal of Systems and Software*, 2014, pp. 48–59.

[8] J. H. Bernardo, D. A. da Costa, and U. Kulesza, "Studying the impact of adopting continuous integration on the delivery time of pull requests," in *ESEC/FSE 2015 Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 805–816.

[9] (2019) Continuous integration–thoughtworks. [Online]. Available: https://www.thoughtworks.com/continuous-integration

[10] E. Laukkanen, M. Paasivaara, and T. Arvonen, "Stakeholder perceptions of the adoption of continuous integration–a case study," in *2015 Agile Conference.* IEEE, 2015, pp. 11–20.

[11] B. Vasilescu, S. V. Schuylenburg, J. Wulms, A. Serebrenik, and M. G. van den Brand, "Continuous integration in a social-coding world: Empirical evidence from github," in *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on IEEE*, 2014, p. 401–405.

[12] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, "Usage, costs, and benefits of continuous integration in open-source projects," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering.* ACM, 2016, pp. 426–437.

[13] G. Gousios, M. Pinzger, and A. van Deursen, “An exploratory study of the pull-based software development model,” in *ICSE 2014 Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 345–355.

[14] M. Manglaviti, E. Coronado-Montoya, K. Gallaba, and S. McIntosh, “An empirical study of the personnel overhead of continuous integration,” in *MSR ’17 Proceedings of the 14th International Conference on Mining Software Repositories*, 2017, pp. 471–474.

[15] A. Miller, “A hundred days of continuous integration,” in *Agile 2008 Conference*, 2008.

[16] D. Ståhl and J. Bosch, “Experienced benefits of continuous integration in industry software product development: A case study,” in *The 2013 IASTED International Conference on Software Engineering*, 2013, p. 736–743.

[17] S. Stolberg, “Enabling agile testing through continuous integration,” in *2009 Agile Conference*, 2009.

[18] D. Parsons, H. Ryu, and R. Lal, “The impact of methods and techniques on outcomes from agile software development projects,” in *Organizational Dynamics of Technology-Based Innovation: Diversifying the Research Agenda*, 2007, pp. 235–249.

[19] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, “Usage, costs, and benefits of continuous integration in open-source projects,” in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering - ASE 2016*, 2016.

[20] Y. Zhao, A. Serebrenik, Y. Zhou, V. Filkov, and B. Vasilescu, “The impact of continuous integration on other software development practices: a large-scale empirical study,” in *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, 2017, p. 60–71.

[21] Y. Yu, H. Wang, V. Filkov, P. Devanbu, and B. Vasilescu, “Wait for it: Determinants of pull request evaluation latency on github,” in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, 2015.

[22] Y. Yu, G. Yin, T. Wang, C. Yang, and H. Wang, “Determinants of pull-based development in the context of continuous integration,” in *Sci. China Inf. Sci.*, 2016.

[23] F. Shull, V. Basili, J. Carver, J. C. Maldonado, G. H. Travassos, M. Mendonça, and S. Fabbri, “Replicating software engineering experiments-addressing the tacit knowledge problem,” in *Proceedings International Symposium on Empirical Software Engineering*, 2002, p. 7–16.

[24] N. Juristo and O. S. Gómez, “Replication of software engineering experiments,” in *Empirical Software Engineering and Verification*, 2012, pp. 60–88.

[25] S. McNeeley and J. J. Warner, “Replication in criminology: A necessary practice,” *European Journal of Criminology*, vol. 12, no. 5, pp. 581–597, 2015.

[26] M. C. Makel, J. A. Plucker, J. Freeman, A. Lombardi, B. Simonsen, and M. Coyne, “Replication of special education research: Necessary but far too rare,” *Remedial and Special Education*, vol. 37, no. 4, pp. 205–212, 2016.

[27] J. W. Schooler, “Metascience could rescue the ‘replication crisis’,” *Nature News*, vol. 515, no. 7525, p. 9, 2014.

[28] V. R. Basili, F. Shull, and F. Lanubile, “Building knowledge through families of experiments,” *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 456–473, 1999.

[29] F. J. Shull, J. C. Carver, S. Vegas, and N. Juristo, “The role of replications in empirical software engineering,” in *Empirical Software Engineering*, 2008, p. 211–218.

[30] T. Karvonen, W. Behutiye, M. Oivo, , and P. Kuvaja, “Systematic literature review on the impacts of agile release engineering practices,” in *Information and Software Technology 86*, 2017, p. 87–100.

[31] G. Gousios, “The ghtorrent dataset and tool suite,” in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR ’13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 233–236. [Online]. Available: http://dl.acm.org/citation.cfm?id=2487085.2487132

[32] D. T. Campbell and T. D. Cook, *Quasi-experimentation: Design & analysis issues for field settings.* Rand McNally College Publishing Company Chicago, 1979.

[33] D. L. Thistlethwaite and D. T. Campbell, “Regression-discontinuity analysis: An alternative to the ex post facto experiment.” *Journal of Educational psychology*, vol. 51, no. 6, p. 309, 1960.

[34] G. W. Imbens and T. Lemieux, “Regression discontinuity designs: A guide to practice,” *Journal of econometrics*, vol. 142, no. 2, pp. 615–635, 2008.

[35] F. E. Harrell, *Regression modeling strategies: with applications to linear models, logistic and ordinal regression, and survival analysis.* Springer, 2015.

[36] D. G. Altman and J. M. Bland, “Detecting skewness from summary information,” *British Medical Journal*, vol. 313, no. 7066, pp. 1200–1201, 1996.

[37] Y. Jiang, B. Adams, and D. M. German, “Will my patch make it? and how fast?: Case study on the linux kernel,” in *Proceedings of the 10th Working Conference on Mining Software Repositories.* IEEE Press, 2013, pp. 101–110.

[38] D. A. da Costa, S. McIntosh, U. Kulesza, and A. E. Hassan, “The impact of switching to a rapid release cycle on the integration delay of addressed issues-an empirical study of the mozilla firefox project,” in *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR).* IEEE, 2016, pp. 374–385.

[39] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, “Selecting empirical methods for software engineering research,” *Guide to advanced empirical software engineering*, pp. 285–311, 2008.

[40] (2014) The case for continuous delivery. [Online]. Available: https://www.thoughtworks.com/insights/blog/case-continuous-delivery