A Study of the Difference in the Productivity in Software Development Before and After the Initial COVID-19 Lockdowns

Tapabrata Tapanlal Mukhopadhyay, B. Tech.

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Future Networked Systems)

Supervisor: Stephen Barrett

August 2022

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Tapabrata Tapanlal Mukhopadhyay

August 19, 2022

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Tapabrata Tapanlal Mukhopadhyay

August 19, 2022

A Study of the Difference in the Productivity in Software Development Before and After the Initial COVID-19 Lockdowns

Tapabrata Tapanlal Mukhopadhyay, Master of Science in Computer Science
University of Dublin, Trinity College, 2022

Supervisor: Stephen Barrett

The onset of the COVID-19 pandemic had global effects on people's lives. It had a huge social, physical, and psychological effect on every aspect of people's lives. Given that the immediate response to the spread of the virus was quite radical, it is not too big a leap to wonder whether it had any effect on productivity. This study attempts to study the change in productivity in software development around the time of the initial COVID-19 lockdowns. The study is conducted on a sample of top GitHub repositories based on star count. The study firstly identifies specific, reasonable, and measurable markers for productivity and studies the trends in their mean and distribution before and after the initial COVID-19 lockdowns. This study also contributes with a reusable Python library for the analysis of the data. The study observes an overall decrease in the productivity after the initial lockdowns. It also observes slight changes in the distribution of productivity before and after the lockdowns. The study finally discusses its limitations in granularity and suggests future expansion for developing a more comprehensive understanding of productivity as a metric.

Acknowledgments

I extend sincerest gratitude to my supervisor, Prof. Stephen Barrett. Thank you for lifting me higher than I thought I could go. I express my deepest thanks to my parents. Their belief in me gives me strength. To my friends, some here in Dublin and some far away in India, I learn from you everyday. I also extend my thanks to Trinity, for giving me an opportunity when I was most in need. Finally, to my kid brother, to whom I dedicate this work. Your unconditional love keeps me going in my darkest hours...and look where it has gotten me.

Tapabrata Tapanlal Mukhopadhyay

University of Dublin, Trinity College

August 2022

Contents

List of Tables

List of Figures

Listings

Chapter 1

Introduction

The COVID-19 pandemic had an unprecedented impact on the world of software development. By the end of March 2020, most governments of countries with a wealth of software developers had imposed lockdowns with a Stringency Index[1] {Hale et al. (2021)} of 70 or higher. It is reasonable to suppose that such sudden and drastic measures must have impacted the world of software development in multivariate ways. As an extension of that argument, it is not too big a leap to propose that one of the aspects of that world which must have been affected by the collective response to the pandemic is the productivity.

This study focuses on measuring the change in productivity and its distribution on a sample of top GitHub repositories. It makes a case for reasonable and specific markers for productivity and measures the change in their mean and distribution around the time of the initial COVID-19 lockdowns.

[1] `https://ig.ft.com/coronavirus-lockdowns/`, Last accessed on: 19 August 2022

1.1 Motivation

1.1.1 The Study of Productivity

The Oxford English Dictionary defines productivity[2] as, "the effectiveness of productive effort, especially in industry, as measured in terms of the rate of output (of goods, products, etc.) per unit of input (of labour, materials, equipment, etc.)." Though the definition is mathematical in nature, correct measurement of productivity hinges on the capability of the said 'outputs' and 'inputs' of being measured. Looking at studies about the perceptions of productivity among software developers {}, it can be quickly discovered how subjective and messy a process it shall be to attempt to form a global consensus about the properties that make up the correct measure of productivity and their respective weights in the equation.

It is easier, and perhaps more useful, to come up with a set of reasonable, measurable and irrefutable markers of productivity. By measuring the changes in such markers and observing the correlations in their deltas, real-world implications and conclusions can be drawn. That matters because, regardless of the extent of the markers' contributions to the function to calculate productivity, those are objective measures and not subjective perceptions of productivity. Additionally, if chosen wisely, those measures could form a significant chunk of the ideal equation to calculate productivity.

1.1.2 The Effects of the Responses to the COVID-19 Pandemic on Productivity

Since the early responses to the COVID-19 pandemic had to be radical, those set to motion a series of social and professional experiments across the world. It is worth studying whether these experiments, for instance: work-from-home, have had any impact on productivity. If they have had an impact, it will be further useful to nail down what kind

[2] `https://www.oxfordlearnersdictionaries.com/definition/english/productivity`, Last accessed on: 19 August 2022

of impact they have had. Such studies shall enable managers and developers to make more informed business decisions in the future. For e.g. which experiments to carry on running beyond the pandemic, what trade-off decisions can be made so that developers feel significantly more fulfilled at their jobs while not sacrificing the overall productivity to dangerous degrees which in-turn may lead to increased longevity in software development careers, etc.

1.2 Research Question

This dissertation attempts to explore a solution to the following question:

Had there been a measurable change in the productivity in software development after the initial COVID-19 lockdowns were imposed?

1.3 Research Objectives

This research focuses on measurable differences in factors that contribute to productivity in software development. The research also attempts to investigate the changes in the distribution of productivity. The goal is to implement these experiments and evaluate the observations in the context of the early lockdowns in response to the COVID-19 pandemic. The expectations are that:

1. The study will elucidate on how the productivity in software development was impacted after the changes that were brought about starting with the initial lockdowns of the COVID-19 pandemic. The study will also provide the software metric industry with a transparent report of the experiment.

2. The study will provide the software development industry with a simple and efficient analysis utilizing terms already familiar to the industry as markers of productivity.

3. The study guarantees no gamification of the analysis by the subjects because it is working with past data only.

4. The study will further the academic conversation, with regards to measuring change in the productivity in software development, to a meaningful degree.

   Note: 'Meaningful' is a subjective term. Hence, to be specific - 'meaning', in the context of this study, shall be derived from instances of further replicable, validated, reproducible, and transparent bodies of work that cite this study to support, criticize, include, or extend its findings.

1.4 Research Scope

1.4.1 Importance of Stating the Research Scope

It is crucial to clearly define the scope of the study due to the following reasons:

1. **Replicability:** If the scope of the study is left unclear, it will be difficult or near-impossible to replicate. That is not helpful because a non-replicable study cannot be validated by re-performing the same set of experiments over the same set of inputs.

2. **Context:** A lack of understanding of the scope of a study may lead to a lack of understanding of the context of the study. It is important to know what data has been used and what has been let go to understand the implications of the inclusion/exclusion. Also, in case of time-related events, the correct understanding of the scope makes it possible to map the data to specific events in time which can reveal correlations between events and behavior of data.

3. **Nature of the Data:** Clearly stating the research scope can immediately reveal the properties, limitations, and constraints of the data being used. For e.g. the size of the sample, information about where the data has been collected from, etc.

1.4.2 Stating the Research Scope and Context

1. In this dissertation, the difference in the productivity in software development is studied by picking a sample of only GitHub repositories.

2. The scope of the study has been limited to only the top 20 GitHub repositories, in terms of star count. The reason star count has been chosen as the metric to pick the top repositories is because starring is a way for users to indicate some kind of positive interest or inclination towards a project and hence is a decent marker for the health and popularity of GitHub repositories {Brisson et al. (2020) and Blincoe et al. (2016)}. Other metrics such as fork count, repository size, follower count, etc. are also fair options and there is a case to be made for each one of them. However, in this study, star count has been used to identify the top repositories.

   Note: Since highest star count has been chosen as the metric to sample repositories, the immediate trade-off caused due to that decision needs to be addressed. The specificity of the criteria might mean the existence of bias in the data in some form or the other. For e.g., maybe repositories of certain ages, sizes, or appeal amass higher star counts.

3. The month of March 2020 has been considered as the marker for the initial lockdowns of COVID-19 for this study. That is because among the top 10 nations by number of GitHub users {Wachs et al. (2022)}, 8 out of 10 countries imposed a lockdown with stringency index 70 or higher on March 2020. The only exceptions being China, where restrictions of such stringency were introduced in February 2020, and Japan, where the restrictions never stringent enough to touch as high a stringency index as 70[3]. One caveat here is that this study does not use March 2020 as merely a proxy for remote work. It is instead a proxy for an amalgamation of all the consequences of having a lockdown with a stringency index of 70.

[3] `https://ig.ft.com/coronavirus-lockdowns/`, Last accessed on: 19 August 2022

4. Datasets could not be generated for 2 out of the 20 chosen GitHub repositories (rank 1: freeCodeCamp[4] and rank 11: tensorflow[5]). This could be due to multiple reasons. The usual suspects are the breadth (large number of commits) and the depth (large number of files in a commit) of the repositories. Those reasons, coupled with the fact that there is an API rate limit of 5,000 requests per hour in GitHub, resulted in failure to acquire the data for the aforementioned repositories[6]. So, the study has been conducted on the rest of the 18 repositories out of the chosen 20.

5. The scope of the data mining for this study has been limited to commit-centric data. It is perfectly legitimate to extend the study to consider other factors, for instance, issues and pull requests. However, due to the time constraint under which this study was carried out, the data mining was limited to commit-centric data.

6. The data from the selected GitHub repositories were grouped in four ways for the study: by month, by author, by repository and month, and by repository and author.

7. In the two groupings that involve month, the study evaluates the difference in productivity by comparing it over the period of 12 months before and after the initial COVID-19 lockdowns. The pre-lockdown range is defined as [March 2019, February 2020]. The post-lockdown range is defined as [March 2020, February 2021].

8. In case of the groupings that involve authors, the scope before and after the initial lockdowns is defined as the number of active authors that had contributed to the repository in that time frame. An author is considered to have been contributed if they had had authored at least one file that had been listed as a modified file in a commit to the repository.

[4] `https://github.com/freeCodeCamp/freeCodeCamp`, Last accessed on: 19 August 2022

[5] `https://github.com/tensorflow/tensorflow`, Last accessed on: 19 August 2022

[6] `https://docs.github.com/en/developers/apps/building-github-apps/rate-limits-for-github-apps`, Last accessed on: 19 August 2022

1.5 Research Challenges

1.5.1 Calculating Productivity

Although productivity is a word that is easy to linguistically define, coming up with a quantitative definition for it is rather a challenge. So much of what is perceived as productive is subjective and based on the context of a project. Addition of n lines of code (LOC) in a week might be perceived as an ideally productive contribution for one project. Whereas, for another project, the benchmark for ideal productivity could be adding 10 times of n LOC in a week.

To tackle this challenge, this study focuses on comparing the markers of productivity, over time, on the same GitHub repositories. Since the repositories are not being changed, it results in a controlled environment of sorts where the goalposts for ideal productivity are not significantly moved because the contexts of having the same repositories are being held onto. So, without having to define what the ideal productivity is for a certain repository, it has been established that the ideal productivity for this study has been fixed over time because we are holding onto the context of the project.

This study does not set out to calculate the ideal productivity for each repository nor does it guarantee a completeness in the selection of markers that influence productivity. Instead, the aim is to observe and compare the changes in the identified measurable markers for productivity. For e.g. commit counts, number of LOC, etc. By comparing these changes in the markers, over time, it can be concluded whether there has or has not been any change in productivity, insofar as it is definable by the selected measurable markers.

1.5.2 Selecting the Measurable Markers for Productivity

There are three challenges regarding the selection of the markers:

1. **Assumption:** There is an element of assumption attached to the markers that

shall be chosen. However, the solution is to make the claims so humble that they are categorically irrefutable. For e.g. everyone can agree that if there have been no new commits at all in a repository, it implies that nothing new has been produced and hence the change in productivity has been zero. As an extension to that logic, it is safe to assume that commit counts do influence productivity. The degree of influence may be big or small but it is irrefutable that it does have an influence.

2. **Measurability:** To eliminate subjectivity and to make it possible to study the changes, the chosen markers for productivity need to be measurable. There exist models that attempt to objectively measure perception-based properties. For e.g., the Delta Maintainability Model (DMM) {di Biase et al. (2019)} is a proposed model for the measurement of maintainability of code. For this study, only markers that are either directly measurable or are measurable through reliably validated models have been selected.

3. **What Not to Consider:** Numerous markers could be picked to study productivity. One challenge is to eliminate potential markers which are relatively or completely insignificant such that there lay a workable set of markers to deal with. The risk posed by this challenge was significantly reduced due to limiting the scope of the data being mined, as mentioned in item 5 of subsection 1.4.2 Stating the Research Scope and Context. From the remaining list of markers, the ones that did not qualify the conditions laid out in the previous two items (item 1 and item 2 in subsection 1.5.2 Selecting the Measurable Markers for Productivity), i.e. those that could not be presumed to have an influence on productivity or those that were not measurable, were eliminated.

1.5.3 Difficulties of Mining GitHub Data

Scores of academics have chosen GitHub as the subject for their studies about productivity or contribution in software development {Das (2019); Saadat et al. (2020); Rehman et al.

(2020); Miyashita et al. (2018); Vadlamani and Baysal (2020); Subramanian et al. (2022)}. Although it is a popular subject, it does come with its set of challenges for data miners:

1. The rate limitation[7] on the amount of user-to-server requests per hour stands as a challenge when mining data from GitHub. It is a challenge in terms of both time taken to mine data and time taken to re-test data mining code after bug fixes.

2. Myriads of projects on GitHub are inactive, have low activity, or are not software development projects {Kalliamvakou et al. (2016)}.

3. Alternate ways of gathering GitHub data, as opposed to mining data through GitHub API, have their sets of assumptions and limitations.

4. Access is restricted to only public repositories which may lead to skewed results and incorrect assumptions in studies.

1.6 Dissertation Outline

- Chapter 1 introduces the reader to the key points of the study. It discusses the motivation behind the study, states the research question and research objectives, elaborates on the scope of the research, lists the research challenges, and finally provides an outline of the text.

- Chapter 2 discusses the state-of-the-art work and literature in the field. It starts off by providing a background of the underlying domain of the study, i.e., Software Metrics. Then, it goes on to make a case for the markers that were chosen to study the core of the subject, i.e., productivity. This chapter then introduces the reader to the related work that has been previously done in the field and how they tie to this study. Finally, the base hypothesis of the study is formulated.

[7] `https://docs.github.com/en/developers/apps/building-github-apps/rate-limits-for-github-apps`, Last accessed on: 19 August 2022

- Chapter 3 offers details about the methods and techniques used in the study to attempt to answer the research question. It begins with a low-resolution overview of the approach. Then it delves deeper into the decisions taken regarding process like selection of markers, data gathering, data clustering, pre-processing, the custom library created for the study, and finally the way the results have been generated.

- Chapter 4 delves into the implementation of the research. The technologies used for the same are described and justified.

- Chapter 5 discusses the findings of the study, discusses the evidence that back those findings, and also ties relevant parallels in findings back to previous literature.

- Chapter 6 concludes the dissertation by highlighting how the research objectives were achieved through the study. It also provides a list of limitations of the study. It then culminates the dissertation by discussing the potential future work.

Chapter 2

State of the Art

This chapter aims to lay out the context in which this study is being conducted. It delves into the works of research that have explored similar issues and discusses their conclusions. It attempts to make a case for the markers that have been chosen to represent productivity, in this study, by looking at previous literature and applying analytical and critical thinking to support the decisions. At the end of the chapter, an informed hypothesis is formed, as the starting point of the research in this paper, based on the literature that has been reviewed in this section.

2.1 Background

To form a deep understanding of the state of the art practices in the software development metrics industry, it is vital to integrate the study with the history of this domain.

2.1.1 Software Engineering Metrics: A Brief History

According to Fenton and Neil (1999), the usage of the term 'software metrics' is somewhat deceptive. The term has come to represent a plethora of measurement activities or properties of code in the field of software development. It started in the 1960s with the measure of Lines of Code (LOC) or Thousands of Lines of Code (KLOC) as a proxy for size of program. Such usage of LOC/KLOC as proxy for program size can be seen in works like Putnam (1978) and Boehm (2001).

The first book meant for software metrics was Gilb (1976). Around that time, aspersions were being cast upon the leaps in logic taken in software metrics' research works such as the simplistic practice of treating LOC as a surrogate for the size of a program or the effort for writing the program. As a result, by the late 1980s, much nuanced models of software metrics were being generated. For e.g. Grady and Caswell (1987), which produced the first and most comprehensive company-wide metrics report, and Basili and Rombach (1988), which simple and objective-driven metrics.

The most cutting-edge work in the field of software metrics, as of now, is Das (2019). The study builds a comprehensive, data-oriented model to calculate the contribution of software developers through raw signals mined from GitHub repositories. The contribution scores generated by the model were also validated against manually generated scores. As demonstrated in this study, the trade-off between simple, goal-oriented heuristics and nuances in interpretation of results is the major adversary that future works in software metrics will have to keep contending with.

2.2 A Case for the Chosen Markers of Productivity

This section introduces the markers chosen to measure productivity in this study and it makes a case for why those markers have been chosen.

Since productivity can be defined as the output produced per unit of input, quantifiable inputs and outputs need to be specified in order to measure any aspect of productivity. Two units of input have been selected for this study: month and author. Month represents a unit of time over which output can be produced and author represents a unit of contributor by whom output can be produced.

Six units of output have been selected as markers for this study: commit count, number of additions, number of deletions, aggregate LOC, aggregate complexity, and aggregate token count. The base argument for all of these outputs to be considered as valid markers is that it can be said for each of them that if they are greater than zero, then the change in overall productivity is non-zero. Hence, it is hard to refute that they have some influence on productivity in software development. Moreover, commit count, number of additions, and number of deletions have been used as metrics for productivity in Bao et al. (2020). LOC has been used as a metric for measuring productivity in Boehm (2001) and Devanbu et al. (1996). Gill and Kemerer (1991) tied cyclomatic complexity[1] {McCabe (1976)} to software maintenance productivity. Lastly, token count is used as a metric because it can be couple with LOC to produce a better idea of the size of the work. LOC is good for providing the breadth of the word but token count provides the depth of the work. For instance, if two programs A and B have the same LOC n, but A has a token count x times that of B, one can conclude that program A is x times more verbose than B. Hence, token count does provide additional idea about the produced output than merely the LOC.

2.3 Related Work in Academia

At its core, this paper is a study of how the onset of the COVID-19 pandemic impacted software development. Several researchers have studied aspects of that problem over the past two years. Some of those studies are relevant to the research question that is being tackled in this study.

Ågren et al. (2022) studied the impact of the involuntary move to remote work and the government-imposed societal restrictions on software engineers working in agile projects. According to the study's findings, the agile practitioners who felt compelled to work remotely exhibited lower levels of productivity. While most of open source projects that like the ones sampled for this study do not follow the agile methodology, it is still worth mentioning Ågren et al. (2022) because it will be interesting to figure out if its observations are paralleled in open source software development or not.

[1] `http://users.csc.calpoly.edu/~jdalbey/206/Lectures/BasisPathTutorial/index.html`, Last accessed on: 19 August 2022

Ford et al. (2021) studied self-reported productivity of software developers at Microsoft and reported findings in a similar vein. It concluded that, by and large, there did not seem to be a dramatic shift in productivity in software development. However, on an individual level, the data suggested larger divides. The developers who enjoyed the liberty that came with working remotely reported higher levels of productivity. Whereas, the ones who missed office reported a dip in their productivity. There are three caveats to note with this study. One is that the study had been carried out in a closed source software development environment as opposed to GitHub's open source environment. Secondly, it might not be useful to do a retrospective study with the subjective markers of productivity because one's perception of their productivity might be prone to more errors if they were to estimate it at a later time in the magnitude of years. So, this kind of study is difficult to replicate or reproduce in the open source world. Finally, the third caveat to this study is that, due to the subjective/perceptive nature of it, the results could well be pointing towards a psychological phenomena that is detached from reality. However, the fact that the findings correlate with other studies like Ågren et al. (2022) suggests that there may be more to it.

Ralph et al. (2020) studied responses from 2000+ software developers across 50+ countries to investigate the change in their productivity and well-being. The study discovered through factor analysis methods that the pandemic did have a net negative effect on both productivity and well-being.

Moving on to studies that deal with allied concepts to productivity, the most comprehensive recent study that delves into contribution is Das (2019). It proposes a heuristics data model to attempt to measure contribution of developers to a project based on 'signals' obtained from the GitHub database. It also lays out some challenges of working with GitHub API, for e.g., a few GitHub API endpoints provide only a subset of the relevant data that has been queried and the rest of the data needs to be laboriously mapped out using data mining techniques, which is also pointed out in Kalliamvakou et al. (2016).

There are several other perils of working with GitHub. Kalliamvakou et al. (2016) does not state commit-centric data related problems but it points out more general problems like every repository may not be a project and many active projects do not use GitHub exclusively as their version control tool. Such are the caveats of studying with GitHub's data and the results of any study that samples data from GitHub should be viewed through this context.

2.4 Formulation of the Hypothesis

Based on the literature review and deductive thinking, the base hypothesis for this study is formulated as:

Hypothesis 0: *The results of the study will demonstrate a change in the productivity in software development, insofar as productivity is defined by the chosen markers, after the initial COVID-19 lockdowns.*

Chapter 3

Methodology

This chapter describes the methods used to attempt answering the research question. It elaborates on the decisions made regarding how the research shall be conducted. Firstly, it lays out an overview of the approach in which section it breaks down the methodology into four essential steps. Then, in the rest of the sections, the five steps have been explained in detail.

3.1 Overview of the Approach

To study the change in productivity, the primary task was to select the output markers of productivity. Once those had been identified, the relevant data needed to be gathered from GitHub. Then, the gathered data was grouped and pre-processed to form relevant context-based clusters of data worth studying. Based on the groups, two forms of results were generated: graphs and tables. Finally, to be able to perform the necessary analysis, a library was created with functions that would aid in analysing the data.

So, overall, the methodology can be broken down into the following four steps:

1. Selecting Productivity Markers

2. Gathering Data

3. Grouping Data

4. Generating Results

5. Generating an Analysis Library

3.2 Selecting Productivity Markers

The first, and perhaps the most important, task that needed to be accomplished to commence the study is to identify the markers or productivity. These markers needed to be simple, reliable, measurable, industrially comprehensible, and reasonable.

The six markers that had been chosen to represent productivity are:

1. **Commit Count:** Number of times changes were committed to a repository in a month or by an author.

2. **Additions:** Number of lines added in a commit. It includes both code and non-code lines. For e.g. lines added to the readme file are also included in this count.

3. **Deletions:** Number of lines deleted in a commit. It includes both code and non-code lines.

4. **Aggregate Lines of Code (LOC):** The aggregate of LOC in a commit.

5. **Aggregate Cyclomatic Complexity:** The aggregate number of linearly independent paths through the modified files in a commit.

6. **Aggregate Token Count:** Tokens are the smallest unit of code like constants, operators, and separators. Although LOC might hint at the breadth of the code, it does not provide great insight about the code-depth. Hence, token count is a decent measure to have alongside LOC as it shines a light on the depth of the code when pitted against the LOC.

The rationale behind choosing the above-mentioned markers has been explained in chapter 2 State of the Art section 2.2 A Case for the Chosen Markers of Productivity.

3.3 Gathering Data

After deciding the markers, the next task at hand was gathering the data. The initial method by which data gathering was to be attempted was decided as using scripts with automated HTTP GET requests to fetch the relevant data from the GitHub REST API Mombach and Valente (2018). At this point, a few of the other popular alternates were considered, the major one among them being GHTorrent {Gousios (2013)}. However, due to GHTorrent data only going back till 2012 {Mombach and Valente (2018)}, GitHub API was chosen over GHTorrent just in case the repositories being looked into were older than 2012.

However, during the implementation phase, it turned out to be incredibly challenging to work with the GitHub REST API. The reason could be lack of experience in working with REST API, the complexity of GitHub REST API requests, or a culmination of both. Due to this, midway through the project, the data gathering process had to see a pivot.

The search for an alternative to both GitHub REST API and previously considered solutions led to PyDriller[1] {Spadini et al. (2018)}. PyDriller is a tool that helps mine GitHub data with minimal code. For the kind of markers chosen for this study, PyDriller was the perfect tool for extraction. The reason PyDriller did not show up in the radar of the first search was because the first search looked into popular solutions, which seemed like a reasonable approach at the time, but did not work out well.

Once PyDriller was discovered, it was easy to mine out the data from the chosen repositories, except in two cases (freeCodeCamp[2] and tensorflow[3]), as explained in subsection 1.4.2 Stating the Research Scope and Context. The main advantage to using

[1] `https://pydriller.readthedocs.io/en/latest/`, Last accessed on: 19 August 2022
[2] `https://github.com/freeCodeCamp/freeCodeCamp`, Last accessed on: 19 August 2022
[3] `https://github.com/tensorflow/tensorflow`, Last accessed on: 19 August 2022

PyDriller is that it does not need to rely on a web connection while fetching the required GitHub data. If a GitHub repository is cloned locally, PyDriller is able to fetch the repository's information from the cloned folder. This was a game changer for the data gathering process in this study as it significantly increased the speed at which data was fetched from the selected repositories.

The initial dataset was created by fetching data about all the files that were modified in commits. So, the dataset had a unique row for each modified file in a commit. This kind of dataset was created for each individual repository. Let's call this type of dataset as **dataset A**.

3.4 Grouping Data

Before pre-processing, decisions had to be made on what kind of clusters of data would be analysed. The following four categories of grouping were identified for analysis:

1. **By month:** To study the change in productivity by comparing the trend in mean and shifts in distribution over 12 months before and 12 months after the initial COVID-19 lockdown.

2. **By author:** To study the change in productivity among authors by comparing the trend in mean output and shifts in distribution among the active authors of the repository, before and after the initial COVID-19 lockdowns.

3. **By repository and month:** To study the change in productivity by month but for each individual repository.

4. **By repository and author:** To study the change in productivity by author but for each individual repository.

3.5 Pre-processing

From dataset A, three more types of datasets were generated, one for analysis by month and one for analysis by author, by applying the following pre-processing steps:

3.5.1 For Analysis by Month

1. **Converting the Datetime field in Dataset A to Year-Month pairs:** This was done to enable analysis 'by month' for all the repositories.

2. **Grouping the data by Year-Month pairs:** After the datetime fields were converted to year-month pairs, they were grouped by year-month. While grouping them by year-month, the count of the unique commit hashes per month were retained as commit count and the sums of the rest of the markers were taken per month.

3. **Filling Out Zeroes for Left-out Year-Month pairs:** There could be cases where there were no activities in a repository in a whole month. So, as the final step of pre-processing, those gaps in data were filled out by adding an entry each for such year-month pairs with all the markers set to zero. This resulted in another dataset. Let's call it **dataset B.**

3.5.2 For Analysis by Author

1. **Ordinal encoding of authors in Dataset A:** This was done to give unique numerical IDs to each author in the Dataset A so they could be easily grouped by their unique ID called the 'author code'.

2. **Grouping the data by author code and dividing the dataset:** After each author was given a unique author code, they were grouped by author code and divided into two different datasets. The first dataset, let's call it **dataset C1**, contained all the entries made before March 2020 (let's call it the intervention), and hence formed a pre-intervention dataset. Whereas, the rest of entries went

into a second dataset, let's call it **dataset C2**, which became the post-intervention dataset. While grouping them, the count of the unique commit hashes per author were retained as commit count and the sums of the rest of the markers were taken per author.

3.6 Analysis Library

For analysis of the data, existing measures of analysis were considered. However, there were no studies of productivity that used uniform techniques to measure change or distribution of productivity over both time-series (year-month) data and categorical (author) data. Therefore, an analysis library with custom functions was created that would work on both types of datasets. Following are the functions in the analysis library:

3.6.1 Get Trend in Mean

This function takes in, as arguments, two dataframes (with equal number of columns) and outputs the trend in the mean of each column in the two dataframes.

The idea is that the two dataframes passed as arguments will contain the pre-intervention and post-intervention data respectively. This function then outputs the trend in mean for all the markers (columns) in the two dataframes. Since it works with the mean of all the columns, the length of the columns need not be the same in the two dataframes for this function to work. Hence, it works with both by-month datasets and by-author datasets. In case of by-month analysis, dataset B is divided into pre and post-intervention subsets by year-month and then the subsets are passed as arguments to this function. In case of by-author analysis, the C1 and C2 datasets are passed as arguments to this function.

Following is an example of how the output should look like:

```
Pre and Post-Covid Mean Comparisons:
commit count :  370  vs  422  ->  Up (14.1%)
```

```
additions :  50513  vs  61162  ->  Up (21.1%)
deletions :  27264  vs  30944  ->  Up (13.5%)
agg loc :  1305  vs  637  ->  Down (51.2%)
agg complexity :  160  vs  100  ->  Down (37.5%)
agg token count :  8580  vs  4195  ->  Down (51.1%)
```

3.6.2 Get Equilibrium Number

This function takes in a list of values as argument and returns a number called the equilibrium number.

Equilibrium Number: It is the percentage of the highest values in the list whose sum is equal to the sum of the rest of the values in the list.

If a list of values of a marker, let's say commit count, is passed from a 'by-month' dataset as an argument to this function and the function returns x. Then, it means that as many commits were made in x% months as were made in the rest of (100-x)% months.

Similarly, if a list of values of a marker, let's again say commit count, is passed from a 'by-author' dataset as an argument to this function and the function returns x. Then, it means that as many commits were made by x% authors as were made by the rest of (100-x)% authors.

The equilibrium number is always >0 and $<=50$. Its utility of it is that it reveals the evenness of productivity distribution. The closer the equilibrium number is to 50, the more fairly the output (marker) was distributed per unit input (month or author).

The change in the equilibrium number before and after the intervention reveals the change in the distribution of productivity.

3.6.3 Get Pareto Number

This function also takes in a list of values as argument and returns a number called the Pareto number.

Pareto Number: It is the percentage of the highest values in the list whose sum is equal to 80% of the sum of all the values in the list.

The utility of the Pareto number is similar to that of the equilibrium number except it tells what percentage of the top producing inputs it took to produce 80% of the outputs.

The Pareto number is always > 0 and $<= 80$. The closer the number is to 80, the more even the distribution of productivity. The closer the number is to 0, the less fairly distributed the productivity is.

Although it does a similar job to the equilibrium number, the Pareto number is being looked into because it is also of academic interest. Numerous natural phenomena play out in the world according to the Pareto principle {Newman (2005)}, which states that roughly 80% of outcomes are produced by 20% of causes. It is one of the most common parallels observed in relation to productivity, in general, in the world. So, it would be interesting to observe whether the Pareto principle plays out for any of the output-input combinations in this context as well.

3.7 Generating Results

Finally, results are generated in two ways:

3.7.1 Graphs

For the 'by-month' datasets, a quarterly (every 3 months) moving average is calculated for all the markers. Then, a year-month vs marker scatter plot is generated with the moving average of that marker represented as a blue line and the intervention month (March 2020) marked with a vertical red line.

The scope of the graphs for the aggregated by-month plots is from March 2019 to January 2022 because those are the maximum start date and minimum end date, respectively, for a repository among the sample set of selected repositories.

No graphs were generated for authors because there are usually too many authors in a medium to big repository and a lot of them contribute rarely. So, it would not serve any meaningful purpose to plot an authors vs marker plot.

3.7.2 Tables

Four tables were generated for the four different groupings as mentioned in section 3.4 Grouping Data. The tables contain the output data from the analysis library listed by each repository and marker (only marker in case of the aggregated tables). The repository-wise tables also note the direction of change in the Equilibrium and Pareto numbers.

Chapter 4

Implementation

This chapter discusses how the study was implemented. It describes the technologies used to realise the methodology.

4.1 Programming Language Used

Python: Python[1] is a powerful, high-level, and efficient programming language with a focus on readability of code. It is also dynamically-typed and interpreted. It is praised for its expansive standard library that supports myriads of tasks.

Python was chosen for this project due to three key reasons:

1. It has brilliant Hypertext Transfer Protocol (HTTP) support.
2. It can get complex tasks done with minimal code.
3. It has great modules which make tasks like working with dataframes, generating plots, and performing analysis easier than in many other programming languages.

Python version used: *Python 3.8.8*

Integrated Development Environment (IDE): *jupyter-notebook : 6.3.0*

[1] `https://www.python.org/`, Last accessed on: 19 August 2022

4.2 Setting Up the Platform

As a first step, the top 20 repositories based on star count were found using an HTTP GET request through Postman[2]. The following GET Request was used (results in top 20 repositories as of 19 August 2022) along with a GitHub bearer token[3] for authorization:

```
https://api.github.com/search/repositories?q=stars:>=138000
```

Then, using the repository links from the result of the above API request, a folder of top repositories based on star count is created with a sub-folder for each repository, ranked by order and the default dataset generation and analysis files and libraries are copied to each repository sub-folder from a template folder. All these steps are performed using a Python script called *top_repos.ipynb*.

The script fetches all the json files stored, which are categorically named, in the ../postman/top_repos_json_files/ folder and creates category-based sub-folders in a new ./top_repos/ directory. Then, it further creates the sub-sub-folders for all the repositories by fetching their links from the json file. So, this script can be used not just to create folders of top repositories based on star count but by any category one wishes by adding the relevant json file to the ../postman/top_repos_json_files/ path.

4.3 Data Gathering

The data gathering has been done using the PyDriller[4] framework available for Python. The steps in data gathering involved are:

[2] `https://www.postman.com/`, Last accessed on: 19 August 2022

[3] `https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token`, Last accessed on: 19 August 2022

[4] `https://pydriller.readthedocs.io/en/latest/`, Last accessed on: 19 August 2022

4.3.1 Cloning GitHub Repository to Local System

In each repository's folder, there is a *git_clone.py* file which is a script that reads the repository's Uniform Resource Locator[5] (URL) from a TXT[6] file with the URL of the repository written into it previously by the *top_repos.ipynb* script. It then clones the repository using the link and the *git* Python module. After the cloning is successful, it produces a message-beep sound to notify the user of the completion.

4.3.2 Using PyDriller to Fetch Data from Cloned Repository

After the repository has been cloned, the notebook file *'1. generate_dataset_modified_files.ipynb'* needs to be run. This program uses PyDriller to fetch the required commit and modified files related data from the cloned repository. The pseudo-code snippet in the Listing 4.1 Fetching Repository Data Using PyDriller demonstrates how the relevant data is fetched from the cloned repository.

```
commit_data = []

from pydriller import Repository

commit_data = [ [
        commit.hash,
        commit.author.name,
        commit.author.email,
        commit.committer_date,
        commit.merge,
        file.filename,
```

[5]https://www.techtarget.com/searchnetworking/definition/URL, Last accessed on: 19 August 2022

[6]https://docs.fileformat.com/word-processing/txt/, Last accessed on: 19 August 2022

```
            file.change_type,
            file.added_lines,
            file.deleted_lines,
            file.nloc,
            file.complexity,
            file.token_count
        ] for commit in Repository(repo_path).traverse_commits()
        for file in commit.modified_files
]
```

Listing 4.1: Fetching Cloned Repository Data Using PyDriller

Then, the program renames the columns of the dataframe to make those comprehensible in English and saves the dataframe as a CSV[7] file in the *datasets* sub-folder of the *repository* folder.

4.4 Data Grouping

After the data is gathered, the *'2. generate_dataset_groupedby_yearmonth'* file should be run. It inserts a new column called *year_month* into the dataframe, read from the CSV file generated in the previous step, by converting the values of the *commit_datetime* column into monthly period type as demonstrated in the below code snippet in Listing 4.2 Inserting Year-Month Column by Converting Datetime Column.

```
df.insert (
    loc=1,
    'year_month',
    pd.to_datetime(df['commit_datetime'],
```

[7] `https://docs.python.org/3/library/csv.html`, Last accessed on: 19 August 2022

```
                                   utc=True).dt.to_period('M') )
df
```

Listing 4.2: Inserting Year-Month Column into the Dataframe by Converting Datetime Data into Year-Month Data

Then, it groups the data in the dataframe by year-month, counting the unique commit hashes every year-month to come up with the commit count for that period, and summing up the rest of the marker values.

It then stores the dataframe into another CSV file in the datasets sub-folder.

4.5 Data Analysis

There are two different Jupyter [8] source files in each repository folder to analyse the data: one provides the analyses results by month (*'3. analyze.ipynb'*) and the other provides analyses results by author (*'4. author_distribution.ipynb'*).

4.5.1 Results by Month

The *'3. analyze.ipynb'* file reads the previously year-month-wise grouped dataframe CSV. If any year-month pair is missing in the file, meaning there was no work produced in that repository that month, then a row is added for the missing year-month with all the markers set to 0 using the *new_zero_row* function. Then, the dataframe is sorted, in ascending order, according to the year-month column so it forms a perfect time series sequence. This dataset is then saved into another CSV file in the *datasets* sub-folder.

For the results, firstly, a moving average column is added for every marker column and a rolling average calculator is used to calculate a moving average with a quarter of the year (or 3 months) as the window. Then, for each marker, a month vs marker graph is plotted. Those graphs consist of a scatter plot of the marker values, a blue line plot

[8] https://jupyter.org/, Last accessed on: 19 August 2022

representing the quarterly moving average of the marker values and a red vertical line denoting the month of March 2020, the point of the initial COVID-19 lockdowns. These graphs are also saved as JPEG[9] files in the *images* sub-folder of the *repository* folder.

Then, the file utilizes the functions created in the analysis library to output the pre and post-intervention comparisons of the means of the markers, the Equilibrium numbers, and the Pareto numbers (section 3.6 Analysis Library).

4.5.2 Results by Author

The *'4. author_distribution.ipynb'* file reads the dataframe from the CSV file produced at the end of the subsection 4.3.2 Using PyDriller to Fetch Data from Cloned Repository. It then inserts a *year_month* column into the dataframe similar by executing the same code as Listing 4.2 Inserting Year-Month Column by Converting Datetime Column.

Then, it ordinal encodes the authors using the *commit_author_email* column as shown in the below pseudo-code in Listing 4.3 Ordinal Encoding of Authors.

```
def get_ordinal_encoding_map(series):
    enc_map = series.unique()
    enc_map = dict(enumerate(enc_map.flatten(), 1))
    enc_map = {val: key for key, val in enc_map.items()}
    enc_map['None'] = 0
    return enc_map


enc_map = get_ordinal_encoding_map(df["commit_author_email"])
df["author_code"] = df["commit_author_email"].replace(enc_map)
```

Listing 4.3: Ordinal Encoding of Authors to Get Unique Author Codes

[9] `https://www.adobe.com/creativecloud/file-types/image/raster/jpeg-file.html`, Last accessed on: 19 August 2022

After getting the author codes, it is ensured that no author codes are set to zero (invalid). Then, the dataframe is divided into pre and post-intervention dataframes based on year-month and they are also grouped by author code to form dataframes as illustrated in Figure 4.1 Dataframe Grouped by Author Code.

	author_code	commit_count	additions	deletions	agg_loc	agg_complexity	agg_token_count
0	1	160	7549	727	692.0	192.0	4511.0
1	2	2	3	3	0.0	0.0	0.0
2	3	1	1	1	0.0	0.0	0.0
3	4	2	4	0	0.0	0.0	0.0
4	5	1	8	8	0.0	0.0	0.0
...	...	...	...	...	...	...	...
77	78	1	1	0	0.0	0.0	0.0
78	79	1	1	1	0.0	0.0	0.0
79	80	1	1	1	85.0	27.0	597.0
80	81	1	2	2	0.0	0.0	0.0
81	82	1	1	1	0.0	0.0	0.0

82 rows × 7 columns

Figure 4.1: Example of Dataframe Grouped by Author Code

Then, the file uses the analysis library functions to output the pre and post-intervention comparisons of the means of the markers, the Equilibrium numbers, and the Pareto numbers (section 3.6 Analysis Library).

4.5.3 Results of Aggregated Datasets

The per-repository datasets that were generated are also clubbed together to analyse the aggregated data by month and by author. The results of this analysis are actually the preliminary results which need to be looked at first before looking into the per-repository analyses. However, it was easier to implement these files in the reverse order, i.e., to generate the per-repository datasets first and then to join them and produce the aggregated results. However, it is recommended that the results be looked into as the aggregated results being the primary results and the per-repository results as a step further in the granularity of analysis.

The aggregated by-month analysis has been implemented in the '*top_repos_by_star _count_analysis.ipynb*' file. The final datasets produced in the subsection 4.5.1 Results by Month are merged to form an aggregated by-month dataframe. Then, the maximum start year-month and minimum end year-month, considering all the repositories in the dataframe, are calculated. The scope of the aggregated dataframe is then reduced to the range within the calculated year-month pair. Then the data is grouped by year-month and the results are derived just like they were derived in subsection 4.5.1 Results by Month.

The aggregated by-author analysis has been implemented in the '*top_repos_by_star_count_author_analysis.ipynb*' file. It merges the datasets produced in the subsection 4.3.2 Using PyDriller to Fetch Data from Cloned Repository. Then, it follows the same steps as subsection 4.5.2 Results by Author to generate the results.

4.6 Implementation of the Analysis Library

The analysis library (*analysis_lib.py*) helps in performing three types of analyses:

4.6.1 Trend in Mean

The trend in mean has been explained in the subsection 3.6.1 Get Trend in Mean. It is implemented using three functions in the analysis library, known as the *calc_and_compare_means()*, the *get_mean()* and the *get_trend()*. The *calc_and_compare_means()* function is the only one that needs to be imported from the analysis library. The below pseudo-code snippet in Listing 4.4 Trend in Mean Snippet demonstrates the implementation of this functionality in the code.

```
def calc_and_compare_means ( df1 , df2 , colStart , colEnd ):
```

```
    print ( 'Pre and Post-Covid Mean Comparisons:' )

    for idx in range(colStart, colEnd):

        pre_mean = get_mean ( df1[df1.columns[idx]] )
        post_mean = get_mean ( df2[df2.columns[idx]] )
        mean_trend = get_trend ( pre_mean, post_mean )
        print ( df1.columns[idx].replace('_',' '), ': ',
            pre_mean, ' vs ', post_mean, ' -> ', mean_trend )
```

Listing 4.4: Trend in Mean Pseudo-code Snippet

Here, the *get_mean()* function returns the mean of a series and the *get_trend()* function returns the trend in the two means as a string. For e.g. 'Down (66.7%)', 'Up (16.5%)', or 'No change'.

4.6.2 Equilibrium Number

The concept of the Equilibrium number has been explained in the subsection 3.6.2 Get Equilibrium Number. It has been implemented in the code as illustrated in the Listing 4.5 Equilibrium Number Snippet.

```
def calc_perc ( idx, length ):

    perc = (idx + 1) / length * 100
    return perc



def get_equilibrium_pc ( lst ):
```

```
    lst.sort ( reverse = True )

    for idx in range ( len(lst) ):
        if ( sum(lst[0:idx+1]) >= sum(lst[idx+1:]) ):
            perc = calc_perc ( idx, len(lst) )
            if perc > 50:
                perc = 50 - ( perc - 50 )

            perc = "{:.1f}".format(perc)
            return perc
```

Listing 4.5: Equilibrium Number Snippet

The function, firstly, sorts the list it takes in as input in descending order. Then it traverses through each index and checks if the sum of all elements until that index >= the sum of the rest of the elements in the list. When the condition is satisfied, it calculates the percentage of the list that has been traversed and returns it. The equilibrium number cannot be greater than 50.

4.6.3 Pareto Number

The concept of the Pareto number has been explained in the subsection 3.6.3 Get Pareto Number. It has been implemented in the code as illustrated in the Listing 4.6 Pareto Number Snippet.

```
def calc_perc ( idx, length ):

    perc = (idx + 1) / length * 100
    return perc
```

```
def get_pareto_pc ( lst ):

    lst.sort ( reverse = True )
    total = sum(lst)

    for idx in range ( len(lst) ):
        if ( sum(lst[0:idx+1]) >= (0.8 * total) ):
            perc = calc_perc ( idx, len(lst) )
            if perc > 80:
                perc = 80

            perc = "{:.1f}".format(perc)
            return perc
```

Listing 4.6: Pareto Number Snippet

Like the equilibrium number function, the Pareto number function also starts with reverse-sorting the list it takes in as input. Then it traverses through each index and checks if the sum of all elements until that index >= 80% of the sum of all the elements in the list. When the condition is satisfied, it calculates the percentage of the list that has been traversed and returns it. The Pareto number cannot be greater than 80.

Chapter 5

Results and Discussion

This chapter lays out the findings of this study, one-by-one, and discusses the evidence that support the findings through the results that have been generated by the implementation. Where appropriate, this chapter also ties the findings back to previous works in literature by drawing relevant parallels.

5.1 Finding #1: Hypothesis 0 is True

Recalling the base hypothesis formulated in section 2.4 Formulation of the Hypothesis, it was stated as:

Hypothesis 0: *The results of the study will demonstrate a change in the productivity in software development, insofar as productivity is defined by the chosen markers, after the initial COVID-19 lockdowns.*

Table 5.1 Aggregated By-Month: Trend in Mean shows the trend in mean, pre and post-intervention, of the value of the all the markers of productivity, aggregated by month. It can be observed from this table that there have been changes in the mean of all the markers of productivity.

The result images in Appendix A Aggregated By-Month Result Images: Year-Month vs Marker also corroborate the evidence provided in Table 5.1 Aggregated By-Month:

Marker	Trend in Mean (Month)
Commit Count	Down (14.2%)
Additions	Down (17.3%)
Deletions	Down (25.4%)
Aggregate LOC	Up (7.0%)
Aggregate Complexity	Down (12.0%)
Aggregate Token Count	Down (0.4%)

Table 5.1: Aggregated By-Month: Trend in Mean

Trend in Mean.

Note: The trend in mean was calculated by comparing the means of the markers in the ranges [March 2019, February 2020] and [March 2020, February 2021].

Table 5.2 Aggregated By-Author: Trend in Mean shows the trend in mean, pre and post-intervention, of the value of the all the markers of productivity, aggregated by author. It can be observed from this table that there have been changes in the mean of all the markers of productivity, except commit count.

Marker	Trend in Mean (Author)
Commit Count	No change
Additions	Down (1.7%)
Deletions	Down (11.3%)
Aggregate LOC	Up (27.2%)
Aggregate Complexity	Up (4.9%)
Aggregate Token Count	Up (18.4%)

Table 5.2: Aggregated By-Author: Trend in Mean

Putting together the evidence shown in Table 5.1 Aggregated By-Month: Trend in Mean, Appendix A Aggregated By-Month Result Images: Year-Month vs Marker and Table 5.2 Aggregated By-Author: Trend in Mean, it can be concluded that there is definitely a change in the productivity, insofar as it is defined by the chosen markers, after the initial COVID-19 lockdowns hit. Hence, Hypothesis 0 is true.

5.2 Finding #2: Overall Productivity Reduced in the Year After the Initial COVID-19 Lockdowns Were Imposed

Observing Table 5.1 Aggregated By-Month: Trend in Mean, it was noted that all the markers of productivity, except LOC, saw significant dips in mean. Therefore, it can be concluded that, overall, there had been a slight dip in the productivity, as defined by the chosen markers, in the year after the initial COVID-19 lockdowns were imposed.

Note: This finding is similar to the primary finding of Ralph et al. (2020), as mentioned in the section 2.3 Related Work in Academia. It is worth noting that in Ralph et al. (2020), the study had been conducted on anonymous participants through a survey and it had used factor analysis to capture change in productivity. In contrast, this study is conducted on past open source software data and the change in productivity is captured by looking at the trends in the means of chosen markers of productivity.

5.3 Finding #3: Productivity Distribution Got Slightly More Even Across Time (Months)

Marker	**Month Eq (pre)**	**Month Eq (post)**
Commit Count	41.7	50.0
Additions	41.7	41.7
Deletions	33.3	41.7
Aggregate LOC	41.7	41.7
Aggregate Complexity	33.3	41.7
Aggregate Token Count	41.7	41.7

Table 5.3: Aggregated By-Month: Equilibrium Numbers, Pre and Post-Intervention

Table 5.3 Aggregated By-Month: Equilibrium Numbers and Table 5.4 Aggregated By-Month: Pareto Numbers, respectively, present the Equilibrium Numbers and Pareto

Marker	Month 80 (pre)	Month 80 (post)
Commit Count	75.0	80.0
Additions	75.0	75.0
Deletions	66.7	75.0
Aggregate LOC	66.7	75.0
Aggregate Complexity	66.7	75.0
Aggregate Token Count	66.7	75.0

Table 5.4: Aggregated By-Month: Pareto Numbers, Pre and Post-Intervention

Numbers for all the markers of productivity grouped by month, before and after the initial COVID-19 lockdowns (the intervention event) were imposed.

It can be observed in Table 5.3 Aggregated By-Month: Equilibrium Numbers that 50% of the equilibrium numbers increase slightly and the rest of the 50% stay as they were. Also, in the Table 5.4 Aggregated By-Month: Pareto Numbers, it can be observed that 5 out of 6 Pareto numbers go up and 1 remains the same.

Hence, due to the observed net rises in the equilibrium and Pareto numbers, it can be concluded that the productivity was slightly more evenly distributed across time in the year that followed the initial COVID-19 lockdowns.

5.4 Finding #4: Productivity Distribution Changed Slightly Among Authors

Marker	Author Eq (pre)	Author Eq (post)
Commit Count	1.0	1.5
Additions	0.5	0.7
Deletions	0.3	0.4
Aggregate LOC	0.2	0.1
Aggregate Complexity	0.2	0.2
Aggregate Token Count	0.2	0.2

Table 5.5: Aggregated By-Author: Equilibrium Numbers, Pre and Post-Intervention

Table 5.5 Aggregated By-Author: Equilibrium Numbers and Table 5.6 Aggregated

Marker	Author 80 (pre)	Author 80 (post)
Commit Count	15.7	14.4
Additions	1.5	2.4
Deletions	1.1	1.5
Aggregate LOC	0.3	0.4
Aggregate Complexity	0.3	0.5
Aggregate Token Count	0.3	0.4

Table 5.6: Aggregated By-Author: Pareto Numbers, Pre and Post-Intervention

By-Author: Pareto Numbers, respectively, present the Equilibrium Numbers and Pareto Numbers for all the markers of productivity grouped by author, before and after the initial COVID-19 lockdowns (the intervention event) were imposed.

It can be observed in both Table 5.5 Aggregated By-Author: Equilibrium Numbers and Table 5.6 Aggregated By-Author: Pareto Numbers that there is only slight changes in the equilibrium and the Pareto numbers pre and post the lockdowns.

Hence, it can be concluded that the productivity distribution changed only slightly among authors before and after the initial COVID-19 lockdowns were imposed.

5.5 Finding #5: Commits are Relatively More Evenly Shared Among Authors Compared to the Rest of the Markers, Both Pre and Post-Intervention

It can be observed in Table 5.6 Aggregated By-Author: Pareto Numbers, that the Pareto numbers among authors for commit count are significantly higher than the Pareto numbers of the rest of the markers. This suggests that, no matter the lockdowns, there remains a high disparity between the distribution of commit counts, than the rest of the markers, among authors. The commit count is shared significantly more evenly than the rest of the markers.

Note:

- This phenomena could be due to GitHub being an open source platform where new authors may come and make relatively small changes to a project while the rare few that know the project inside-out remain the highest contributors in terms of code, documentation, and complexity.

- There is a slight decrease in the disparity between commit count distribution and rest of the markers' distribution post-intervention due to Pareto number of commit counts decreasing and the Pareto number of the rest of the markers increasing.

5.6 Finding #6: Commit Count Among Authors the Only Marker to Come Close to Following the Pareto Principle

As mentioned in subsection 3.6.3 Get Pareto Number, the Pareto principle follows the 80-20 rule. As observed in Table 5.6 Aggregated By-Author: Pareto Numbers, the only marker that comes close to following the Pareto Principle is the commit count.

Pre-lockdowns, 15.7% of the authors made 80% of the commits and post-lockdowns, 14.4% of the authors have made 80% of the commits. Both the Pareto numbers for commit counts are the closest among the markers to being in line with the Pareto Principle.

Chapter 6

Conclusion

This chapter culminates the dissertation. It is divided into three parts. The first part discusses how the research objectives mentioned in section 1.3 Research Objectives have been met. The second part discusses the limitations and the disadvantages to the approaches undertaken in this study. The final section lists some future work prospects based on the study conducted in this project.

6.1 Objective Assessment

6.1.1 Elucidate on the Impact of the Initial COVID-19 Lockdowns on Productivity

Research Objective: *The study will elucidate on how the productivity in software development was impacted after the changes that were brought about starting with the initial lockdowns of the COVID-19 pandemic. The study will also provide the software metric industry with a transparent report of the experiment.*

The markers for productivity that were chosen in this study, have been chosen with sound logic and evidential backing based on previous work. The research question was clearly answered when the base hypothesis was proven based on the results of the study.

Also, the report has been written in as transparent a fashion as possible, to the best of the author's ability.

6.1.2 Study Provides Simple and Efficient Analysis to Software Development Industry

Research Objective: *The study will provide the software development industry with a simple and efficient analysis utilizing terms already familiar to the industry as markers of productivity.*

The methodology used has been straightforward and the terms used in the project are all according to the software development industry standards. The only non-industrial academic contribution this study attempts to make is by the introduction of the concept of the Pareto number, which is based on the Pareto principle. Still, although it may not be directly industrially relevant, it does offer utility by providing a decent way of measuring distribution in productivity.

6.1.3 No Gamification of Analysis

Research Objective: *The study guarantees no gamification of the analysis by the subjects because it is working with past data only.*

As promised in the objective, the study works with past data from 18 GitHub repositories.

6.1.4 Furthering the Conversation of Measuring Change in Productivity

Research Objective: *he study will further the academic conversation, with regards to measuring change in the productivity in software development, to a meaningful degree.*

Through the study, a case has been made for a simple and efficient way of studying the delta in productivity. However, it in no way, shape, or form is a complete answer to the puzzle of productivity. Whether this study has achieved this research objective remains to be seen.

6.2 Limitations

1. This study is prone to possible **selection bias** because of it studies top GitHub repositories based on star count and maybe there are certain biased properties that make a repository more likely to have a higher star count in GitHub.

2. This study has a very limited **sample size**. It would definitely be beneficial to carry out this study over a much bigger sample size. For e.g., 100 repositories.

3. Though a decent case is made for the markers used to measure change in productivity, there is a decent case out there for a lot of other markers of productivity too. Hence, the study is **far off from providing a complete picture of productivity**.

4. It was a **struggle to find similar work** in the literature. Although there have been research that study change in productivity, a lot of them are done in closed source settings and numerous of them rely on a subjective idea of perceived productivity.

5. It was **difficult to work with the GitHub REST API** to fetch data. A lot of days were lost trying to make mine data from the API.

6. The PyDriller framework was finally used to mine data. So, any **limitations of PyDriller were inherited**.

7. Results were gathered for each individual repository being studied but the **findings are based on the aggregate results only**.

8. There could be a lot of holes in the analysis of the observations due to **lack of expertise** of the author in this field.

9. Change in productivity is studied around a pivot-point or an intervention. It does not **account for the many nuances in situations** that kept arising post initial COVID-19 lockdowns. For e.g. changing severity of lockdowns, discovery of vaccines, introduction of vaccines to the market, etc.

10. The **Equilibrium Number and Pareto Number functions work better with larger lists** than with smaller lists. Hence, they work better with repositories that have more instances of commits.

6.3 Future Work

1. Many more markers of productivity can be identified and added to the study. For e.g., the PyDriller framework provides the Delta Maintainability score that measures risk of lines of code. This could be added into the mix to knock down productivity scores of high-risk lines of code.

2. The study could be done at a much more granular level. Instead of taking an average point of intervention, the study could be furthered to investigate the effects of lockdowns in various countries for example.

3. The simple addition of non-commit-centric data to the mix (for e.g. issues and pull requests) will make for a more detailed and perhaps better study.

Bibliography

Bao, L., Li, T., Xia, X., Zhu, K., Li, H., and Yang, X. (2020). How does working from home affect developer productivity? - A case study of baidu during COVID-19 pandemic. *CoRR*, abs/2005.13167.

Basili, V. R. and Rombach, H. D. (1988). The tame project: towards improvement-oriented software environments. *IEEE Transactions on Software Engineering*, 14(6):758–773.

Blincoe, K., Sheoran, J., Goggins, S., Petakovic, E., and Damian, D. (2016). Understanding the popular users: Following, affiliation influence and leadership on github. *Information and Software Technology*, 70:30–39.

Boehm, B. W. (2001). *Software Engineering Economics*, pages 99–150. Springer Berlin Heidelberg, Berlin, Heidelberg.

Brisson, S., Noei, E., and Lyons, K. (2020). We are family: Analyzing communication in github software repositories and their forks. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 59–69.

Das, S. (2019). Sociometrics in software engineering : Measuring the contribution of a software engineer in a software project through github. Master's thesis, University of Dublin, Trinity College.

Devanbu, P., Karstu, S., Melo, W., and Thomas, W. (1996). Analytical and empirical evaluation of software reuse metrics.

di Biase, M., Rastogi, A., Bruntink, M., and van Deursen, A. (2019). The delta maintainability model: Measuring maintainability of fine-grained code changes. In *2019 IEEE/ACM International Conference on Technical Debt (TechDebt)*, pages 113–122.

Fenton, N. E. and Neil, M. (1999). Software metrics: successes, failures and new directions. *Journal of Systems and Software*, 47(2):149–157.

Ford, D., Storey, M.-A., Zimmermann, T., Bird, C., Jaffe, S., Maddila, C., Butler, J. L., Houck, B., and Nagappan, N. (2021). A tale of two cities: Software developers working from home during the covid-19 pandemic. *ACM Transactions on Software Engineering and Methodology*, 31(2).

Gilb, T. (1976). *Software Metrics*. Chartwell-Bratt.

Gill, G. and Kemerer, C. (1991). Cyclomatic complexity density and software maintenance productivity. *IEEE Transactions on Software Engineering*, 17(12):1284–1288.

Gousios, G. (2013). The ghtorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 233–236, Piscataway, NJ, USA. IEEE Press.

Grady, R. B. and Caswell, D. L. (1987). *Software Metrics: Establishing a Company-Wide Program*. Prentice Hall.

Hale, T., Angrist, N., Goldszmidt, R., Kira, B., Petherick, A., Phillips, T., Webster, S., Cameron-Blake, E., Hallas, L., Majumdar, S., and Tatlow, H. (2021). A global panel database of pandemic policies (oxford covid-19 government response tracker). *Nature Human Behaviour*, 5(4).

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., and Damian, D. (2016). An in-depth study of the promises and perils of mining github. *Empirical Software Engineering*, 21(5).

McCabe, T. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320.

Miyashita, Y., Hazeyama, A., Hashiura, H., Goto, M., and Hirasawa, S. (2018). A visualization system of the contribution of learners in software development pbl using github. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 695–696.

Mombach, T. and Valente, M. T. (2018). Github rest api vs ghtorrent vs github archive: A comparative study.

Newman, M. E. J. (2005). Power laws, pareto distributions and zipf's law. *Contemporary Physics*, 46(5):323–351.

Putnam, L. H. (1978). A general empirical solution to the macro software sizing and estimating problem. *IEEE Transactions on Software Engineering*, SE-4(4):345–361.

Ralph, P., Baltes, S., Adisaputri, G., Torkar, R., Kovalenko, V., Kalinowski, M., Novielli, N., Yoo, S., Devroey, X., Xin, T., Zhou, M., Turhan, B., Hoda, R., Hata, H., Robles, G., Fard, A. M., and Alkadhi, R. (2020). Pandemic programming. *Empirical Software Engineering*, 25(6).

Rehman, I., Wang, D., Kula, R. G., Ishio, T., and Matsumoto, K. (2020). Newcomer candidate: Characterizing contributions of a novice developer to github. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 855–855.

Saadat, S., Newton, O. B., Sukthankar, G., and Fiore, S. M. (2020). Analyzing the productivity of github teams based on formation phase activity. In *2020 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, pages 169–176.

Spadini, D., Aniche, M., and Bacchelli, A. (2018). *PyDriller: Python Framework for Mining Software Repositories.* Association for Computing Machinery.

Subramanian, V. N., Rehman, I., Nagappan, M., and Kula, R. G. (2022). Analyzing first contributions on github: What do newcomers do? *IEEE Software*, 39(1):93–101.

Vadlamani, S. L. and Baysal, O. (2020). Studying software developer expertise and contributions in stack overflow and github. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 312–323.

Wachs, J., Nitecki, M., Schueller, W., and Polleres, A. (2022). The geography of open source software: Evidence from github. *Technological Forecasting and Social Change*, 176:121478.

Ågren, P., Knoph, E., and Berntsson Svensson, R. (2022). Agile software development one year into the covid-19 pandemic - empirical software engineering. *Empir Software Eng*.

Appendix A

Aggregated By-Month Result Images: Year-Month vs Marker

Following are the year-month vs marker result images of the group 'By-Month':

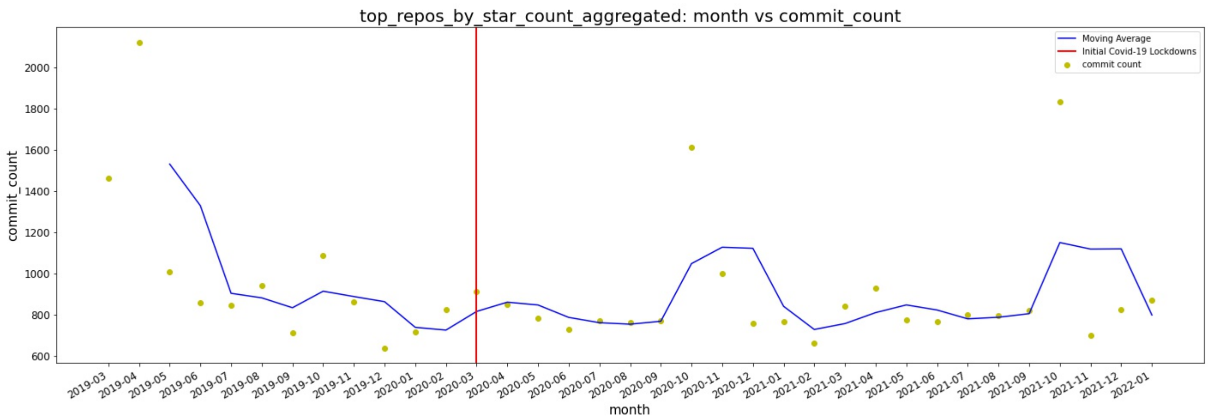

Figure A.1: Aggregated By-Month: Year-Month vs Commit Count

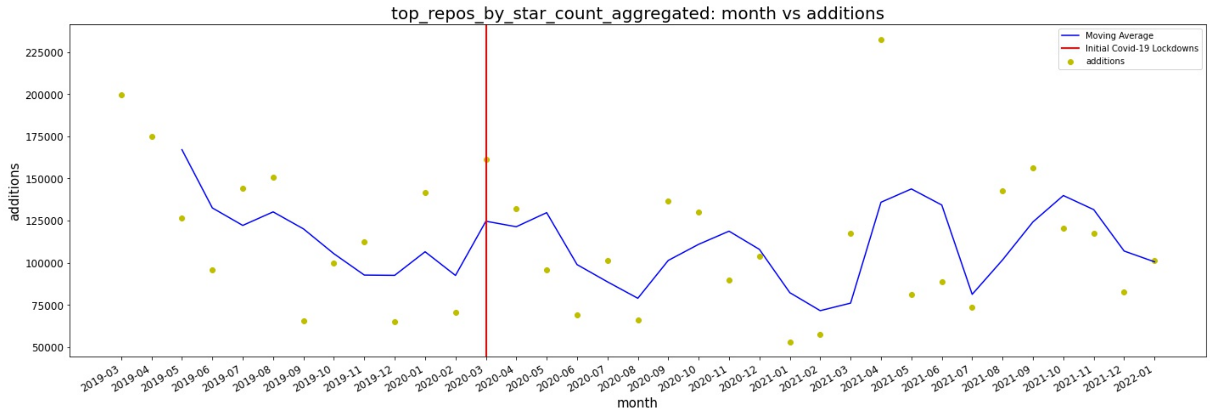

Figure A.2: Aggregated By-Month: Year-Month vs Additions

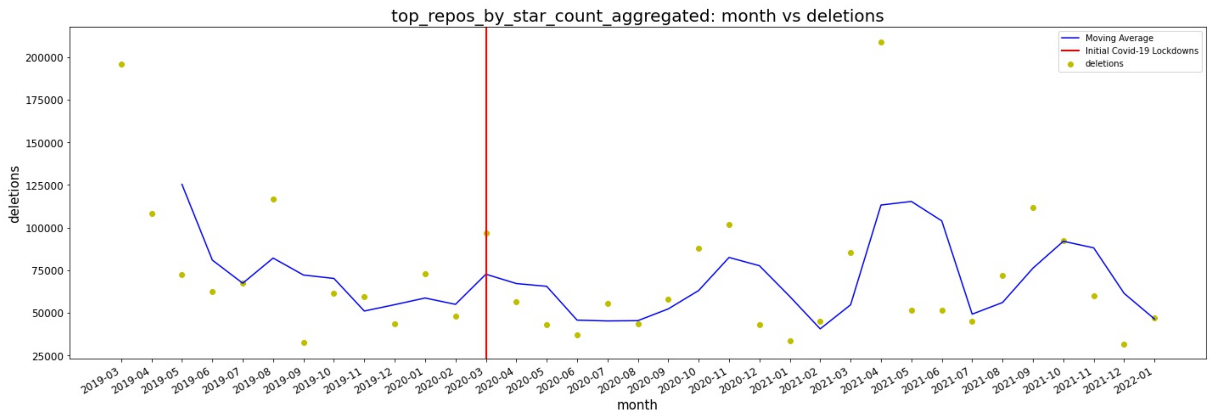

Figure A.3: Aggregated By-Month: Year-Month vs Deletions

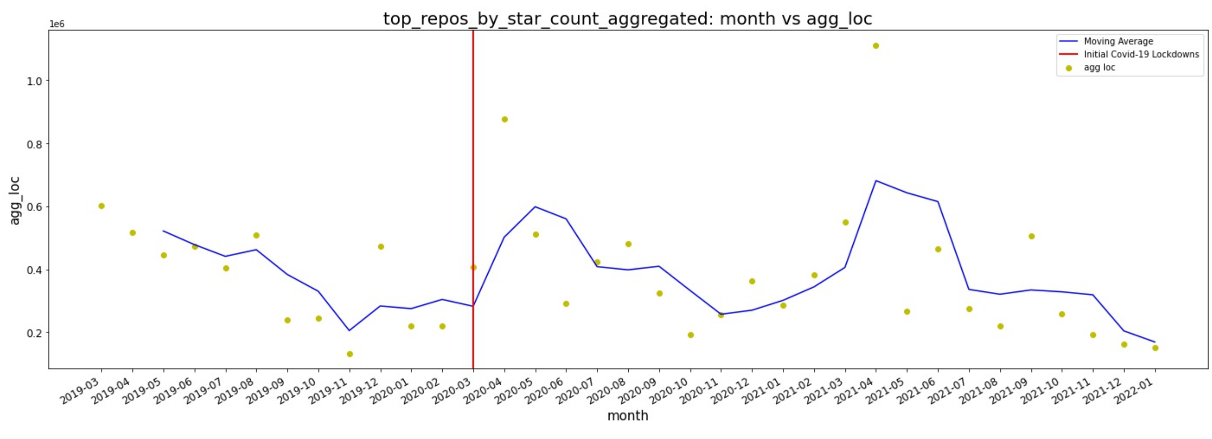

Figure A.4: Aggregated By-Month: Year-Month vs LOC

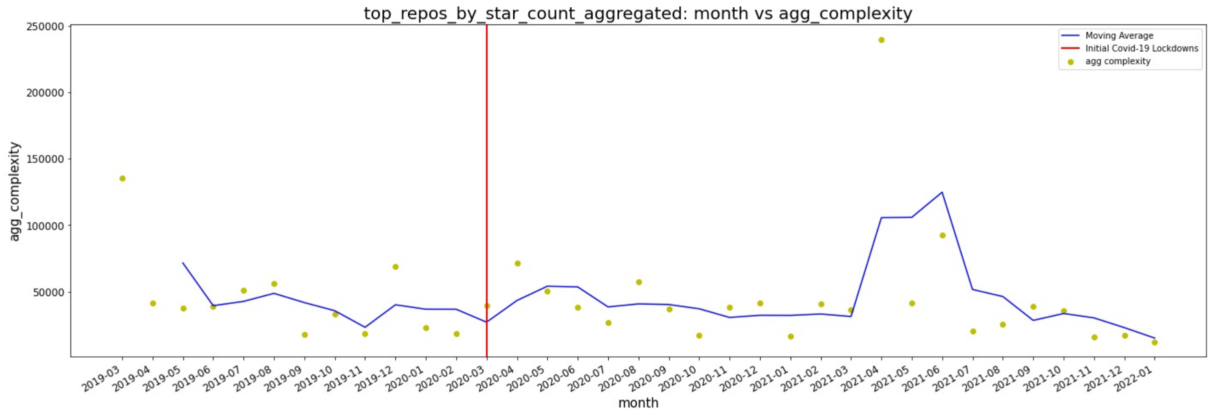

Figure A.5: Aggregated By-Month: Year-Month vs Complexity

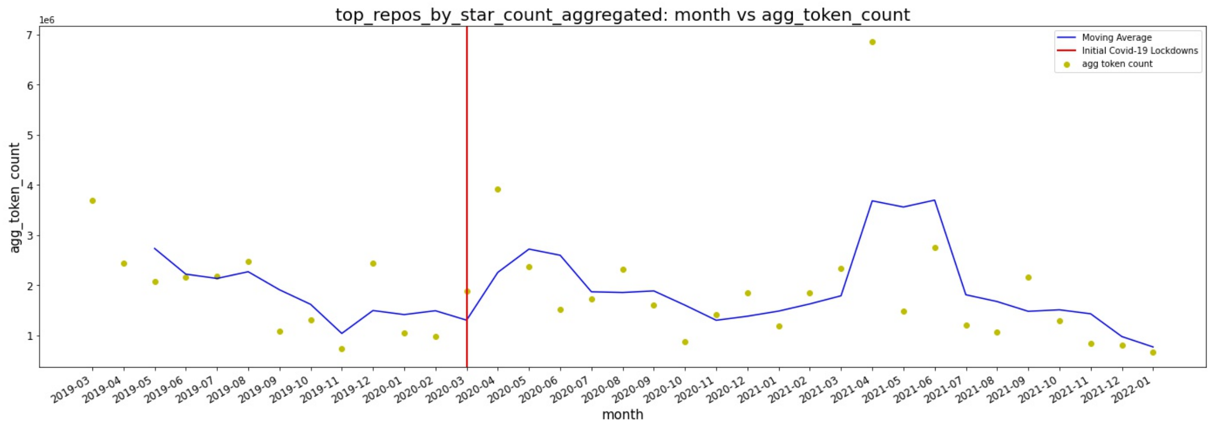

Figure A.6: Aggregated By-Month: Year-Month vs Token Count

Appendix B

Code and Results

All code and results can be found on:

`https://github.com/mukhopat/CS7CS5-202122-Dissertation/`

All the code can be found in the directory called ‘Sprint-1’.

All the results can be found in the directory called ‘all-results’.