OPEN SOURCE SOFTWARE GITHUB ECOSYSTEM: A SEM APPROACH

Thesis submitted by

MOHAMMAD AZEEZ ABDULHASSAN ALSHOMALI

B.A., Al-Mustansiryia University, College of Education, Iraq

M.A., Al-Mustansiryia University, College of Science, Iraq

for the degree of Doctor of Philosophy

in the College of Business, Law & Governance

James Cook University

November 2018

Statement of Access

I, the undersigned, author of this work, understand that James Cook University will make this thesis available for use within the University Library and, via the Digital Theses Network, for use elsewhere.

I understand that, as an unpublished work, a thesis has significant protection under the Copyright Act and;

I do not wish to place any further restriction on access to this work.

_____ _____

Signature Date

Statement of Sources

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institution of tertiary education. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references given.

Signature	Date

Electronic Copy

I, the undersigned, the author of this work, declare that the electronic copy of this thesis provided to the James Cook University Library, is an accurate copy of the print thesis submitted, within the limits of the technology available.

_____ _____

Signature Date

Statement on Contribution of Others

The following contributions of others to the intellectual, physical and written work of this research higher degree thesis are gratefully acknowledged

Stipend support:	Al-Mustansiryia University, Collage of Science, Iraq
Honoraria:	James Cook University, College of Business, Law & Governance
	James Cook University Graduate Research School (GRS)
Supervisor:	Dr. Jason Holdsworth
	Professor John Hamilton
	Dr. Singwhat Lee
Statistical support:	Professor John Hamilton
	Dr. Singwhat Lee
Editorial Assistance:	Dr. Jason Holdsworth
	Professor John Hamilton
	Dr. Singwhat Lee

ACKNOWLEDGEMENTS

First and foremost, praises and thanks to ALLAH (God), the almighty, merciful and passionate, for His blessings throughout my research work to complete the research successfully.

I owe my deep sense of gratitude to my primary advisor Dr. Jason Holdsworth for his help, advice, and support. I offer my sincerest gratitude to my secondary advisors Professor John Hamilton, and Dr SingWhat Tee, for their advice, support, enthusiasm and faith in me. I attribute the level of my PhD thesis to their encouragement and effort.

Nobody has been more important to me in the pursuit of this repo than the members of my family. I would like to thank my brothers; whose love and guidance are with me in whatever I pursue. Most importantly, I wish to thank my loving and supportive wife and my three wonderful children, Arwa, Ali and Mustafa, who provide unending inspiration.

Special thanks to Rafid Al-Hallaf, Mohamed Nazir, Karim Haj Hashemi and Dr. Mustafa Al-Hassani for their help and support since the first day I arrived in Australia. I extended my thanks to my fellow PhD students, academics, and administrative staff at Cairns Campus of James Cook University for their variable support.

ABSTRACT

Open source software (OSS) is a collaborative effort. Getting affordable high-quality software with less probability of errors or fails is not far away. Thousands of open-source projects (termed repos) are alternatives to proprietary software development. More than two-thirds of companies are contributing to open source. Open source technologies like OpenStack, Docker and KVM are being used to build the next generation of digital infrastructure. An iconic example of OSS is 'GitHub' - a successful social site. GitHub is a hosting platform that host repositories (repos) based on the Git version control system.

GitHub is a knowledge-based workspace. It has several features that facilitate user communication and work integration. Through this thesis I employ data extracted from GitHub, and seek to better understand the OSS ecosystem, and to what extent each of its deployed elements affects the successful development of the OSS ecosystem. In addition, I investigate a repo's growth over different time periods to test the changing behavior of the repo. From our observations developers do not follow one development methodology when developing, and growing their project, and such developers tend to cherry-pick from differing available software methodologies.

GitHub API remains the main OSS location engaged to extract the metadata for this thesis's research. This extraction process is time-consuming - due to restrictive access limitations (even with authentication). I apply Structure Equation Modelling (termed SEM) to investigate the relative path relationships between the GitHub- deployed OSS

elements, and I determine the path strength contributions of each element to determine the OSS repo's activity level.

SEM is a multivariate statistical analysis technique used to analyze structural relationships. This technique is the combination of factor analysis and multiple regression analysis. It is used to analyze the structural relationship between measured variables and/or latent constructs.

This thesis bridges the research gap around longitude OSS studies. It engages large sample-size OSS repo metadata sets, data-quality control, and multiple programming language comparisons. Querying GitHub is not direct (nor simple) yet querying for all valid repos remains important - as sometimes illegal, or unrepresentative outlier repos (which may even be quite popular) do arise, and these then need to be removed from each initial OSS's language-specific metadata set.

Eight top GitHub programming languages, (selected as the most forked repos) are separately engaged in this thesis's research. This thesis observes these eight metadata sets of GitHub repos. Over time, it measures the different repo contributions of the deployed elements of each metadata set.

The number of stars-provided to the repo delivers a weaker contribution to its software development processes. Sometimes forks work against the repo's progress by generating very minor negative total effects into its commit (activity) level, and by sometimes diluting the focus of the repo's software development strategies. Here, a

fork may generate new ideas, create a new repo, and then draw some original repo developers off into this new software development direction, thus retarding the original repo's commit (activity) level progression.

Multiple intermittent and minor version releases exert lesser GitHub JavaScript repo commit (or activity) changes because they often involve only slight OSS improvements, and because they only require minimal commit/commits contributions. More commit(s) also bring more changes to documentation, and again the GitHub OSS repo's commit (activity) level rises.

There are both direct and indirect drivers of the repo's OSS activity. Pulls and commits are the strongest drivers. This suggests creating higher levels of pull requests is likely a preferred prime target consideration for the repo creator's core team of developers.

This study offers a big data direction for future work. It allows for the deployment of more sophisticated statistical comparison techniques. It offers further indications around the internal and broad relationships that likely exist between GitHub's OSS big data. Its data extraction ideas suggest a link through to business/consumer consumption, and possibly how these may be connected using improved repo search algorithms that release individual business value components.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

LIST OF APPENDICES

LIST OF ABBREVIATIONS AND ACRONYMS

AGFI	Adjusted Goodness of Fit Index
AMOS	AMOS is statistical software
AMOS	Analysis of a Moment Structures (software)
ANOVA	Analysis of Variance
API	Application Program Interface
API	Application program interface
ASD	Agile Software Development
AVE	Average Variance Extracted
C#	C Sharp (programing language)
C++	C Object-Oriented (programing language)
CCQM	Component Quality Model
CFA	Confirmatory Factor Analysis
CFI	Comparative Fit Index
CMC	Computer-Mediated Communications
COTS	Commercial of the Shelf
CSS	Cascading Style Sheets (programing language)
CSV	Comma Separate Value
DF	Degree of Freedom
EFA	Exploratory Factor Analysis
GFI	Goodness of Fit Index
IFI	Incremental Fit Index
JAVA	General-purpose computer programming language
JavaScript	Programing Language
JS	Java Script programing language
MATLAB	Matrix Laboratory (Programing Language)
NATO	The North Atlantic Treaty Organization
NFI	Normative Fit Index
OSS	Open Source Software
OSSD	Open Source Software Development
OSSECO	Open Source Software Ecosystem
PHP	Hypertext Preprocessor (programing language)
POSSD	Phase Role Skill Responsibility
Repo	Repository
REST	REST- Representational State Transfer, (internet protocol)
RMSEA	Root Mean Square Error of Approximation
ROC	Rate of Change
Ruby	Programing language
SECO	Software Eco System
SECO	Software Eco-System
SEM	Structural Equation Modelling
SPSS	Statistical Package for the Social Science (software)
SVN	Subversion
TK	Tukey Kramer (statistical test)
TLI	Tucker-Lewis Index
VCS	Version Control System

CHAPTER 1

INTRODUCTION

1.1 Introduction

Software is a collection of executable programming code, connected libraries, and support documentation (Cosentino & Cabot, 2017; Haigh, 2011). The process of developing a software product includes initial development of software, maintenance and updates, until the desired software product is developed, which also satisfies the expected requirements. Software and hardware developments affect the way we live.

Today, world depends heavily on software. Software development methodologies attracts researchers to research in that field. The first conference to widely discuss this issue was the NATO conference in 1968 (Randell, 1996). The conference investigated software modelling approaches, and sequential methodology emerged as a key early software development methodology (Papadopoulos, 2015).

Sequential methodology divided software development into consecutive stage requirements, analysis study, design, implementation and maintenance (Atoum & Bong, 2015). Such traditional software development methodologies have been deployed to overcome software problems, and to deliver satisfying end-user solutions. Here, the software should suitably meet the end-user requirements and be deemed to be sufficiently correct, robust, flexible, reusable and efficient (Atoum & Bong, 2015).

Traditional software development does possess advantages, but the resultant systems do not become available to end-users until the development process is complete

(Tachizawa & Pozo, 2012). This approach has embedded time-related risks, which can lead to budget over-runs. Another risk lies in the lack of flexibility typically required particularly when end-users change their requirements during, or after the sequential software development stages (Papadopoulos, 2015).

In 2001 The Agile Software Development (ASD) methodology was introduced to overcome the drawbacks of traditional methods. ASD is easy to understand and implement. It requires the customer to be involved during all stages of software development. It offers flexibility in requirement changing (Amir et al., 2013). Although ASD is considered a good solution for building software that satisfies customers (Dingsøyr et al., 2012), it still can exceed its estimated timeline and budget - particularly where there is a lack of reusability, extensive testing and documentation sometimes fails. This encourages researchers to search for new and better software models (Shah et al., 2012).

Software is ubiquitous, cellular devices, shopping and selling, banking and finance, construction and logistics and most governmental or learning institutes each utilize specific purpose-built software (Qassimi & Rusu, 2015). Today some traditional software fails (Papadopoulos, 2015) because it has not transformed to ASD formats, or because it could no longer deliver the solution required.

The speed and scope of software development remains important because of its increasing need within new applications such as: eBusiness, social media, manufacturing, transport, and finance (Brunetti & Heuser, 2014). Thus, software

researchers typically develop or modify existing models to build quality software within an affordable budget, and within a chosen timeframe.

Open source software (OSS) represents a different methodology of software that is built and distributed through the Internet (Lin & Serebrenik, 2017). OSS refers to software that is developed, tested, or improved through public collaboration. It is distributed with the idea that it must be shared with others, ensuring an open future collaboration. OSS repos are typically built, maintained and tested by a teamed network of global and geographically-distributed open-source community volunteer programmers (Bose & Thakur, 2013; Olson & Rosacker, 2012).

OSS offers an array of co-operative global testing environments where code is dynamically tested, de-bugged and fixed cooperatively between developers (Chou & He, 2011; Sarka & Ipsen, 2017). NOKIA, IBM and Microsoft enlist OSSD within their product develop cycles (Diaz et al., 2009). Although a firms' involvement advances the ranking of OSS repos, it could lower project quality, because firms put corporate constraints to OSSD practices (Hertel & Herrmann, 2003). But research into OSSD cycles remains scant (Jones, 2014).

OSS offers a variety of benefits to the developers and business. In addition to cost reduction commercial companies benefit from contributing to OSS by building their innovation capability, as well as selling emergent complementary services (Andersen-Gott et al., 2012). Learning is the main motivation towards contributing in OSS developments. Altruism, ideology learning, popularity, self-efficacy, and enjoyment

motivates many programmers, users, and businesses towards continuance contributions into repos (Lakhani & Von, 2003; Capra, et al., 2011; Choi & Yi, 2015). OSS is not used alone when designing entire software repos because it is by nature chaotic (Siau & Tian, 2013).

Although software (prosperity or open) is well constructed, occasionally program failures can cost lives and/or money (Boin & Fishbacher-Smith, 2011; Coelho & Valente, 2017). The important gap in OSS research is how to sidestep software failures.

As can be seen from the above, software has developed over the last few years very rapidly from traditional development methodology through to Agile methodology. Once OSS launched no general methodology existed. Instead there exists many software related publications that discuss development methodology for OSS and for OSSD ecosystem platforms such as GitHub (Kalliamvakou et al., 2016). GitHub represents the largest OSS development industry (Bose & Thakur, 2013). To better understand the current trends (and advancements) in OSS, the OSS ecosystem is investigated in next section.

1.2 OSS Ecosystem

The concept of ecosystem transferred from biology to the social world explaining the evolutionary nature of interrelations among different individuals, their innovative activities, and their environment (Papaioannou et al., 2009). As there is a natural ecosystem, so to there is an Industrial ecosystem of business application types. There

is also a software ecosystem of programmable language application types. This software ecosystem is now termed 'SECO' (Manikas & Hansen, 2013).

SECO is a field of increasing importance in research and industry. There is no standard analytical model for it (Manikas & Hansen, 2013). Mens et al. (2014) define and compare software ecosystems against natural (biological) ecosystems. They also draw a representation of software against natural ecosystems - as illustrated in Figure 1-1.

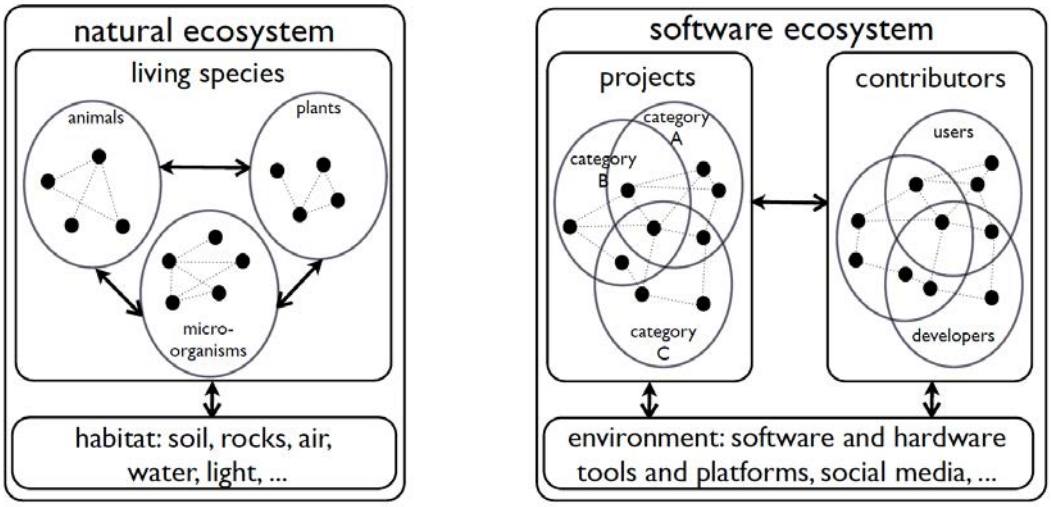


Figure 1-1: Natural versus software ecosystem suggested by Mens, et al. (2014).

Natural ecosystems are sustainable, according to Mens et al. (2014). Sustainability is also a desired characteristic for SECO. Figure 1-1 can be viewed as a comparison between the two ecosystems. Living species within a natural system can be compared to repos in a software ecosystem, whilst Habitat in a natural ecosystem is comparable to resources in a software environment (hardware and software resources). The ecosystem could be interpreted differently by considering projects as part of the environment, and as contributors - equivalent to living species in natural ecosystem.

Mens et al. (2014) suggests Figure 1-1 can represent both a SECO, and an OSS ecosystem (OSSECO).

SECO can be readily deployed as a tool to analyze OSS ecosystems - as it may have strategic, and/or technical, and/or economic advantages. In other words, SECO can help in understanding the different resources across which an OSS may be operating (Franco-Bedoya et al., 2017).

The OSS resources needed can be considered within an OSS ecosystem context as traditional and/or non-traditional (Manikas & Hansen, 2013; Song et al., 2014; Wang et al., 2016; Franco-Bedoya, et al., 2017). Traditional resources directly deal-with (and influence) the OSS, while non-traditional resources indirectly affect the OSS. Figure 1-2 illustrates the subsets of both forms of these resources.

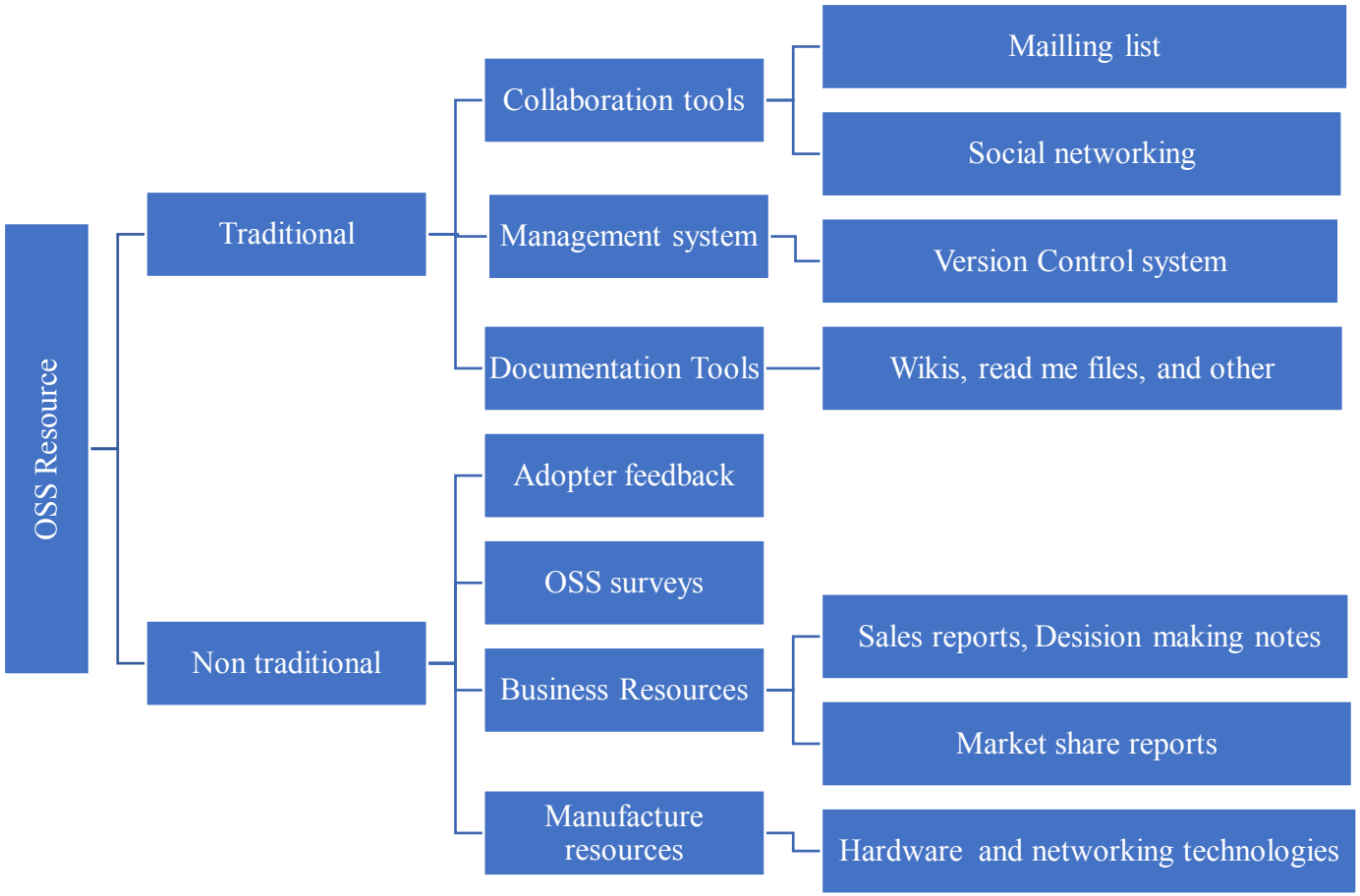


Figure 1-2: Categorisation of OSS resources.

As suggested in Figure 1-2, business resources such as: sales reports, decision-making notes, expert interviews, can directly affect the OSS ecosystem, and vice versa. For example, Amazon has turned almost every successful open source repo into a commercially available, and well-managed service. This has encouraged many developers to join OSS communities like GitHub.

There are many definitions for an OSS ecosystem, and each represents different points of view. Song, et al. (2016) define an OSS ecosystem through technical connectivity between repos, and/or coding, and/or graphical perspectives. According to Song, et al., (2016), interactions and associations collected from varieties of data (and sources), create a new OSS ecosystem. Song, et al. (216) states that, online posts are the foundation element when analyzing an evolution across an OSS development.

"A software ecosystem comprises a set of business, project, and activities that function as a single unit, instead of each participating activity acting individually." This definition is offered in Kilamo et al. (2012). Franco-Bedoya et al. (2017), define a software ecosystem as "one placed in a heterogeneous environment, whose border is a set of niche players, and the keystone player is an OSS community around a set of related repos, and within an open-source (common) platform" (Franco-Bedoya et al., 2017).

1.3 GitHub

GitHub is now the world's largest code host collection of OSS development repos (Gousios et al., 2014). GitHub is an online version-control system used by online open

source software developers (OSSDs) ranging from: professionals to students, from major software companies to small Indie (independent) developers. GitHub provides an environment for developers to share their work, as well as offering its environment for others to use, adapt, and get help when seeking to advance or improve their work (Lanubile et al., 2010).

GitHub repos are diverse in: format, repo-size, development-cycle-stage, releases-count, change-frequency, and changeability. GitHub houses over 20M users and 57M repos (Sharma et al., 2017). It draws worldwide crowd-sourced coding contributors - each with unique individual levels of expertise, into an environment that allows the adding of valuable inclusions into its large number of ongoing software development repos (Tsay et al.,2014b).

GitHub simplifies social coding by providing a web interface to each repo, and the administration tools needed for repo collaboration. GitHub Members can follow each other, obtain updates for repos, rate each other's work and communicate (publicly or privately). Important terms used in GitHub include: pull-request, fork, and merge. A developer creates a repo. Its content is organized into branches, a “master” branch represents the “production code”. Other branches are used for repo contributors to experiment with new features, and for the restructuring of existing features.

The repo evolves over time - from the addition and deletion of content, primarily source code, resource files and documentation. Its changes are tracked via commits - a set of additions and deletions to the content. A developer can “clone” a repo. The

developer gets a complete copy of the content - which they can use for their own purposes. Also, they can ‘fork’ a repo to get a complete copy of the content placed into a new repo - that they own, A fork (or sometimes called branch), is a repo that has been copied from one member's account to another member's account. Forks, and branches, allow a developer to make modifications without affecting the original code. This might represent: (1) a fracture in the ecosystem, or (2) a way for contributors from the original repo to work more independently / safely away from the original content - GitHub uses the Git SVN tool - the tool’s workflow is complex and error-prone (SVN is abbreviation of ‘Subversion’, Subversion is an open-source version control system that is typically used to manage the collections of files that make up software repos.), or (3) a way for a non-contributor of the original repo to make suggestions and to prove whether their ideas are useful. These non-contributors might even wish to become part of the original repo team.

A pull request (a commit that is “merged” into the repo only after approval by the contributors) is the mechanism of suggesting changes/improvements for content. If the developer (who forked a repo) would like to share the modifications he made, then he can send a pull request back to the owner of the original repo. If, after reviewing them, the original owner wishes to pull these modifications into the repo, he can accept and merge these modifications with the original repo. The originating repo development occurs via commits, and branching. by the repo originator, and contributors - who have been added after their fork-and-pull-request process are both accepted into the repo (Lanubile et al., 2010).

GitHub provides social networking tools for developers to communicate, discuss and reason about pull request. It provides many statistics to track all this information (via GitHub API). GitHub enables the creation of large complex interconnected ecosystems of developers. It remains easy for a developer to diagram the possible relationships between the repo, and how it collaborates with other developers) (Arora et al., 2017).

GitHub repos offers a trackable, integrated, time-spanned, workflow of user-delivered, repo contributions. But, the datamining tools used in GitHub remain slightly different from those used in other OSS developments (Kalliamvakou et al., 2016). GitHub repos have a faster growth rate when compared to other rival OSS communities.

The growth and success of GitHub OSS development communities can be viewed as a social activity across contributing developers. This creates a 'herd behavior,' and with growth, repo leadership becomes increasingly important (Hu et al., 2016). GitHub attracts software developers, testers, star coders, coders, social media watchers, and other small solutions pull providers. GitHub repos are easily deployed, follow clear guidelines, engage suitable languages, and are readily utilized to improve the existing OSS development version (Chatziasimidis & Stamelos, 2015).

Coding additions / deletions occur through a series of commits by repo collaborators that update a software codebase. Collaborating and external developers providing pull-request merged commits, are first reviewed and tested by other repos collaborators before their repo code is merged into the main repo codebase. These collaborators are

usually a core team of developers for this repo. Thus, the repo's creator and its core team of collaborators, can be thought of as the ongoing guardians of repo quality (Yu et al., 2014a). The activeness of a repo's creator in handling pull-requests also influences the extent of pull-request activities by the overall ecosystem (Aggarwal et al., 2014).

1.4 GitHub ecosystem

Motivated by Men's et al. (2014) ecosystem, Figure 1-3 illustrate the GitHub ecosystem. This ecosystem supports Men's representation. Basically, there are three elements for GitHub ecosystem, the first one represents GitHub repo with all required Hardware, Software, humans and repos. As illustrated in Figure 1-3, human elements play a major role across: GitHub core developers, across active developers (who may also be a member of the core developer team), across passive developers (who may be followers, watchers or anybody who has no direct influence on the repos but still show an interest), and finally across users who fork and may use the system without participation.

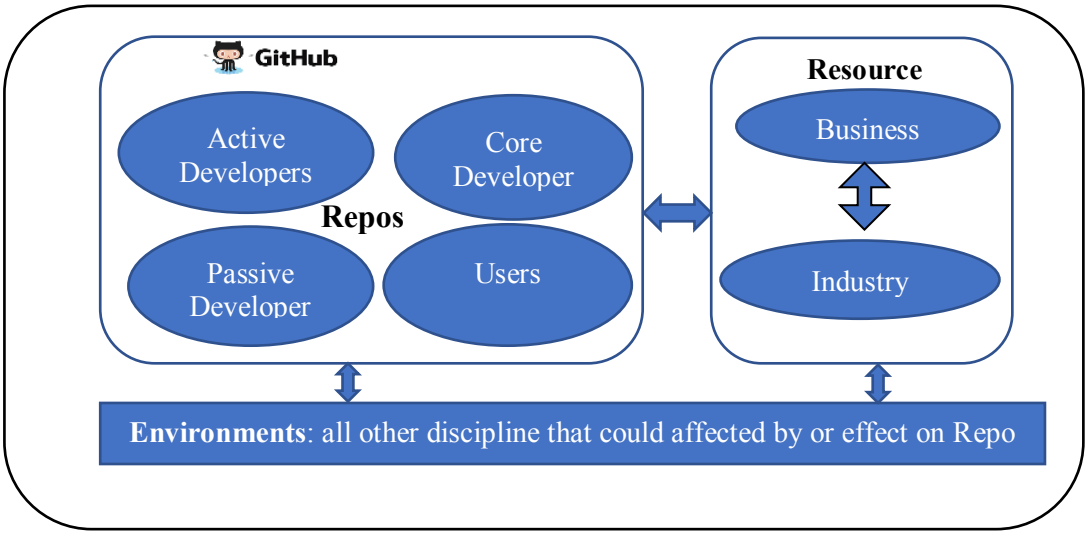


Figure 1-3: GitHub software development ecosystem framework.

The second GitHub ecosystem element is the businesses and the industry that may exert some direct, or indirect effect(s) on the GitHub repo. Google (Android) and Facebook are two examples of businesses that have affected a GitHub repo advancement in hardware manufacturing (industry) - encouraging developers to build software that use the capabilities of the new hardware version - such as in the 'Smartphones' industry.

The third GitHub ecosystem elements is the environment. This represents any other factors that affect GitHub – such as: cultural awareness, confidentiality, language support facilities among developers (communication language and translation).

GitHub repos are diverse in: format, repo-size, development-cycle-stage, releases-count, change-frequency, change-degree, forks, watchers, and contributor-skills (Aggarwal et al., 2014). Such potentially diverse repo variations can also complicate repo comparisons.

When comparing relationships within and around GitHub repos Aggarwal et al. (2014) and Cosentino, et al. (2017) further divide different repos. Their specific categories include: (1) popularity delivering higher/consistent documentation or (2) library repos needing less documentation. Over time, documentation quality improves - especially in larger repos, and as responders (reporters or assignees) become more experienced (Cosentino, et al., 2017; Xavier et al., 2014). Thus, comparative longitudinal GitHub studies remain complex.

From time-to-time GitHub's repo ecosystem may suffer developer and knowledge losses which can retard the repo's software development. For example, a specific developer may choose to externally clone the repo's master branch, and then create a unique (and/or alternate) software development pathway outside the original repo's ecosystem. This loss of developer capabilities likely negatively impacts the repo's development, and it may move other developers away from the original repo, and into following this unique alternative development pathway.

1.5 Research Gap, Questions and Objectives

There are many challenges facing OSS: lack of long-term study, OSS team diversity and the absence of general view of the constructs affecting repos activity (Squire, 2017) are some of these challenges, a review of more OSS challenges presented in Chapter 2. There is little clear identification (and/or standardization) for open SECO. Most studies relating to OSS indicate the lack of longitude studies (West & Gallagher, 2006; Cosentino & Cabot, 2016; Kalliamvakou et al., 2016), and the maintenance of community interest in repos helps in repos evolving and surviving (Cosentino & Cabot, 2016). Previous research seldom measured the influence of OSS features on survival and success. Research in SECO is still in its infancy, and research efforts remain on a slowly increasing trajectory.

Most researchers emphasize the need for more effort in this area of research. Studying OSS repos and the analyzing of the elements affecting those repos today represents a significant step towards better understanding SECO and particularly OSSECO, and

this is the focus of this research. GitHub represents a large OSS community. It provides a big data source-bank for studying and understanding SECO.

In this research I utilize the GitHub ecosystem and use SEM (structural equation modelling) to find the elements that affect the repo's survival. This thesis engages 1600 GitHub repos across eight widely-used, but different programming languages. It then observes repo changes over time and determines the elements that influence a repo's success. This empirical research represents a multi-case study towards modeling the GitHub OSS ecosystem. It attempts to relate, and build, the contributing elements into a systematic model - that influences the survival and successes of the OSS ecosystem.

This thesis answers the following research questions:

RQ1: What elements are present in the GitHub OSS ecosystem?

RQ2: Do programming languages show different models in the GitHub OSS ecosystem?

RQ3: What relationships exist between each element when affecting the commits in the GitHub OSS ecosystem?

RQ4: How does each element influence the GitHub OSS ecosystem?

The objective of this thesis is to capture the big picture around OSS, which is the OSSECO, and to then understand the GitHub ecosystem. This involves understanding the elements that may increase or decrease software activity, and it encourages more user participation. This thesis studies the behavior of repos over times. It determines

good practice for GitHub repo survival. Finally, it determines the criteria used to achieve added performance when stakeholders (especially OSSDs) engage with GitHub (Munaiah et al., 2017).

1.6 Thesis Layout

In addition to this Chapter, which introduces open source software (OSS), the ecosystem and the general framework for GitHub ecosystem, Chapter two presents the literature review - generally classifying research efforts in GitHub as: OSS developments, GitHub measurements, and GitHub challenges and trends.

Chapter three displays two studies using two phases: a pilot study, followed by a series of SEM models each representing one of the eight most popular GitHub languages. To understand GitHub trends, the pilot study deploys 30 repos – ten for each of the three top programming languages used in GitHub (JavaScript, Java and Python). SEM structural path modelling deploys 200 cases for each of eight programming languages (JavaScript, Java, Python, C#, C++, CSS, PHP and Ruby) - thus studying 1600 GitHub repos. The methodology used in both case studies specifies in this Chapter.

Chapter four displays the pilot study and SEM case study results obtained during this two phases study. This Chapter also analyses to what extent each element (or construct modelled under SEM) affects the GitHub ecosystem.

Chapter five provides a discussion of the results obtained in Chapter four. Insights, implications and limitations of these Chapter four SEM models are considered. Chapter six then provide the study's conclusions and ideas for future related studies.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

As discussed in Chapter 1, GitHub is an online version control system used by developers around the world (Gousios & Spinellis, 2012). It currently supports approximately 26 million developers and hosts over 57 million repositories (Sharma et al., 2017). The number of GitHub repositories is growing rapidly compared to other online version control systems (Yu et al., 2014b). GitHub developers include professional developers from the largest (to smallest) software companies, independent developers working on open source software repos, and novice developers working on student repos.

The common terminology for a repository on GitHub is a **repo**. A GitHub repo is an organized collection of content such as source code, multimedia resources and supporting documentation. A ***commit*** represents a set of changes (additions and deletions) to the content of a repo. A 'series of commits' captures how a repo evolves over time. Hence, a repo is the embodiment of a **software ecosystem**.

The developer who creates a repo is known as the repo **creator**. Other developers, known as ***contributors***, are given access to the repo content by the creator. The creator and contributors directly impact the evolution of the repo by adding commits. For example, a contributor might add a commit that solves a problem within the technical capabilities of that contributor (Zhu et al., 2014).

When a repo is created its content is organised into **branches**. The ***master branch*** is a folder within the repo that contains the production content. Other ***developmental branches*** are used by the repo creator and contributors as a place to experiment with new content or refactor existing content.

A non-contributing developer who is external to a repo might ***fork*** it, giving them a complete copy of the repo content and then place it into a new repo that is independently owned by that developer. Forking is a way for original repo contributors to work independently and safely, away from the original content. However, a forked repo also has the potential to draw popularity and interest away from the original repo. Occasionally, this forking can affect the growth of ongoing contributions into the original repo.

When the content of a developmental branch is deemed ready it gets ***merged*** back into the master branch of the repo by the creator of a contributor. Similarly, when the work done on a forked repo is believed ready by its developers, a ***pull*** (a pull request) gets created that represents a potential commit that is mergeable back into the original repo. Before accepting a merge, its review process takes place, allowing the original repo creator and contributors to rationalize the proposed changes. Hence, the pull is either accepted or rejected. If accepted, the merge allows the external developer to become a contributor of the original repo.

If a developer (contributor or not) perceives a problem with repo content, they create an ***issue***. This enables a process whereby the repo creator and contributors rationalise

the issue and mitigate it if necessary. It should be noted that some submissions are not real issues (Bissyandé et al., 2013a). Notice developers sometimes mistakenly create issues that only help requests or act as advice seeking requests.

When a developer ***clones*** a repo, this gives them a complete copy of the repo content without necessarily being an active part of that repo. Unfortunately, GitHub statistics do not track information about cloning. Figure 2-1 summarizes the various aspects of a GitHub repo.

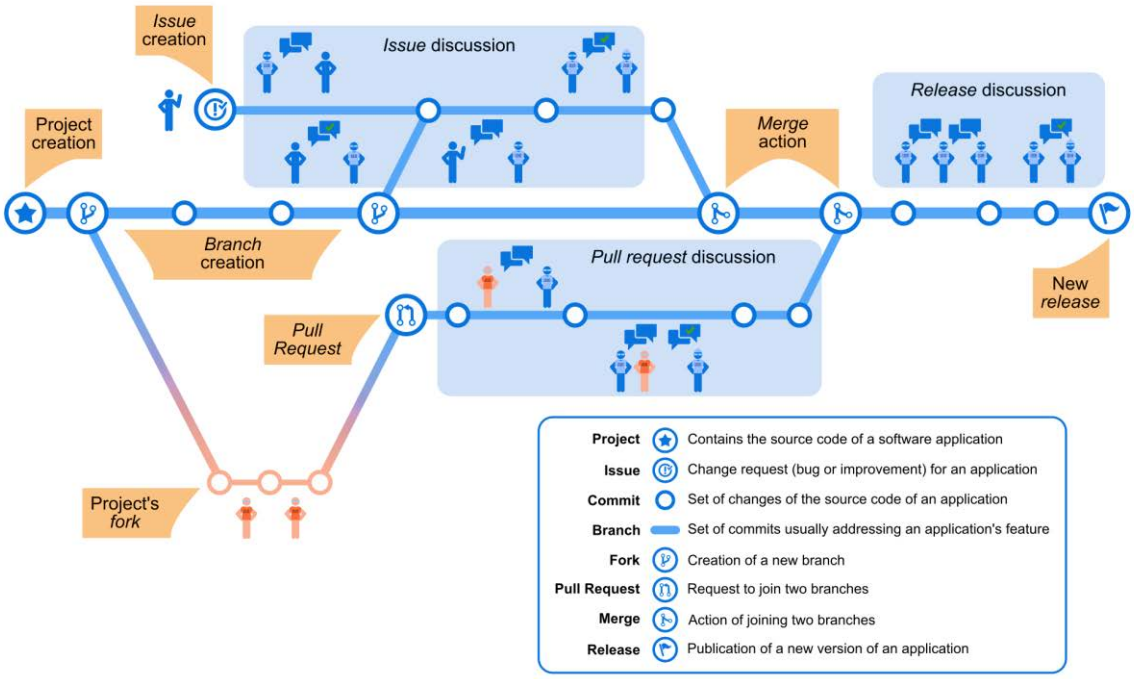


Figure 2-1: Various aspects of a GitHub repo
(From: https://livablesoftware.com/development-process-in-github-basic-infographic/)

2.2 Understanding the GitHub Ecosystem

GitHub itself can be thought of as a massive software ecosystem. GitHub enables and fosters developer collaboration around the world through the creation of repos. For

example, a single developer might choose to create their own repos at the same time as contributing to repos created by other developers. This is important, as it provides developers with an ability to gain technical experience through collaboration, and an ability to build meaningful professional and social relationships in a community of like-minded developers (Casalnuovo et al., 2015).

Overall, the GitHub ecosystem supports developer collaboration by providing social media that provide a range of information about repos (both descriptive and statistical) and the relationships between repos. Developers use this information to discover community-wide popular repos, as well as personally interesting repos. Moreover, this information also encourages developers to get involved in repo issue discussions and/or pull request reviews (Arora et al., 2017).

GitHub provides a freely available repository search engine tool that includes a web REST API. Many third-party web apps utilize the REST API to discover repositories on GitHub (Bello-Orgaz et al., 2016). The API generates data in terms of the information (elements) about repos (Onoue et al., 2013).

GitHub repos vary by the amount and kind of collaborative activity. Such variation depends primarily on the number of commits (Yu et al., 2014b). In addition, pull-requests (both successful and unsuccessful) indicate how a repo evolves over time. Successful pull-requests are merged into the repo - thus adding to the activity level of that repo (Xavier et al., 2014). Figure 2-2 classifies and groups repo developers into different types.

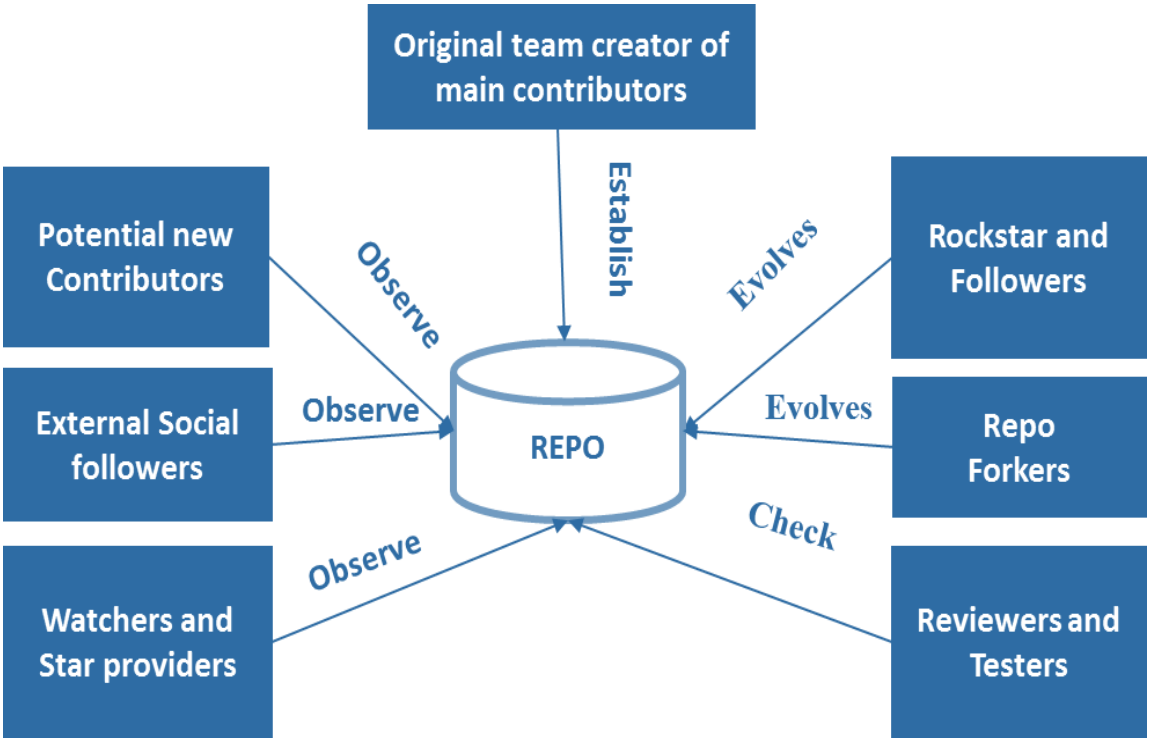


Figure 2-2: Classification of Repository developers.

Rockstars are an important repo contributor whose popularity brings into the repo additional skilled developers. These additional developers often follow the rockstar's lead, and typically generate pull-request activity within the repo (Lee et al., 2013). The presence of Rockstar likely results in an increased repo popularity, generally along with enhanced repo outcomes (Ma et al., 2016). Developers who generate high-quality commits may become recognized as a Rockstar.

The ***fork-repository-clone*** developers are another indication of the repo's popularity. The more forks a repo has, the more likely the repository is recommended, and the higher is the chance to increase the activity of potential new code contributions into the repo (Zhu et al., 2014). Forks sometimes generate strong changes in direction, new

features, better implementation approaches, or even a different version of the existing repo, whilst still keeping their vision around the original repo (Ma et al., 2016).

Reviewers/testers discuss, assess, and recommend each contributor's merging (or rejection) into the repo. When reviewers are specifically assigned, the review or testing process becomes shorter and more effective (Yu et al., 2014a).

A ***watcher/star***-provider receives notifications of any event (commits, pull-requests, and issues) arising within the repo and on GitHub's social media (Ma et al., 2016; Sheoran et al., 2014). It is also common to see popular repos where coding activities are seen to be successful as being 'starred' extensively, and experiencing higher commit frequencies (Cosentino et al., 2017). Watchers tend to contribute to popularity with their external activities on social media, and other digital community forums.

External ***social-followers*** track the actions of other coding developers of good reputation (Luo et al., 2015). Marlow et al. (2013) note GitHub's external social-follower, and reviewer/tester, and watcher/star groups each contribute transparency into a repo (Luo et al., 2015). They also bring additional social considerations, and their social actions can contribute towards the repo's popularity. Potential new contributors can be drawn into a GitHub repo by:

- Adding to current promotional activities;
- Adding to social media, and/or Twitter, and/or Wiki awareness campaigns;
- Following others;
- Adding a piece of personal coding; and
- Sourcing aspects that support a personal area of interest.

2.3 Previous research studies

This Section explores researchers' efforts across three logically-interconnected areas of SECO and particularly OSSECO interest - OSS development methods used by the developer, GitHub components, and mining GitHub (and challenges).

2.3.1 OSS development methods

OSS is not used alone when designing entire software repos because it is by nature disorganised, and this presents risks. Siau & Tian. (2013) developed a theoretical OSSD model which transformed OSSD from a disorganised approach into a semi-organised relational approach. They maintained the OSSD dynamics and developed a Phase Role Skill Responsibility model. This approach deployed Grounded Theory. It is not yet implemented practically only theoretical, and it is still not risk-free. Similarly, Al-Tarawneh et al. (2013) investigate the existing commercial on the shelf (COTS) software and consider its benefits and drawbacks.

They then establish a Component Quality Model for selecting and evaluating existing COTS software. This research was extended by Gandomani et al. (2013). They presented a systematic literature review on the relationship between ASD and OSS. They find a relationship between ASD and OSS exists. However, this relationship remains unconfirmed beyond simple case study experiences. They show the Agile Development methodology (ASD) method and OSSD were related and to date the integration of these two remains unconfirmed - because no successful case studies have emerged, and only a few successful occurrences have emerged (Misra & Singh, 2015; Arora, 2016; Nurdiani et al., 2016).

Understanding the influence of agility in OSS was investigated by Da Silva et al. (2016). The study is ongoing, and the researchers want to measure to what degree ASD applies in OSS releases. The community of developers is the key element of OSS, and its members are crucially motivated to maintain and increase the size of their community (Bahamdain, 2015).

Syeed et al., (2014) link OSS development successes to the volume of GitHub repo community users being deployed. On the other hand, agile team trends suggest the number of developers is optimally 5-9 developers (Williams, 2012). Although more community user numbers deliver more coding changes, Ye and Kishida (2003) found learning to be a key motivational driver in attracting additional software developers.

Van (2016) states there is no standardization for life-cycle shape for collaboration networks in OSS ecosystems. Also, he finds that external factors (such as public holidays) and internal factors (such as software vision) additionally influence collaboration.

Studying GitHub repos, and assessing best OSS practice is increasing understanding around OSS development within GitHub repo communities (Kalliamvakou et al., 2016). This includes learning from past permutations. The success of a project in GitHub helps OSS developers to understand factors that could make the distribution projects success (Hebig et al. 2016; Cosentino et al. 2017).

The literature review above (Kalliamvakou et al., 2016, Williams, 2012,Misra & Singh, 2015; Arora, 2016; Nurdiani et al., 2016, Hebig et al. 2016; Cosentino et al. 2017) suggests OSS is an important approach to software development. OSS provides a low-cost and effective solution for software development. As the OSS development community increases, problems such as poor documentation likely decrease. Hence, by putting documentation regulation inside its developer community domain, it is possible to iteratively advance a repo.

An OSS community's information flows can engender motivational strategies between participant members. GitHub seems to be the best solution for OSSD methods - as it facilitates the collaborative effort by providing tools and platforms for social connection and project development. I expect from the literature that GitHub developers cherry-pick development methods that incorporates ASD and traditional methods.

2.3.2 GitHub Repo Measures

There are measurable elements that directly or indirectly may influence the success of GitHub repos. These components play a central role (according to literature) in repo popularity. Some literature defines repo popularity based on GitHub measurable elements such as stars, forks, watchers, and contributors. Others try to understand GitHub repo classifications using topic modelling. This Section presents relevant literature.

Social media provides an ecosystem for OSSD. Developers use social introductions, as well as other interactions on different platforms (such as Twitter and Facebook) to engage with each other and with GitHub (Wu et al., 2014). For long-term contributions, the presence of past social bonds between developers may not be enough, thus, additional measures may be needed to encourage developer preservation (Casey, 2015). According to Blinco et al. (2016), increased numbers of project contributors will increase project popularity (such as stars and watchers).

Follower and social commentary approaches engage more potential contributors into their chosen GitHub repo. Popular contributors other than rockstars influence their followers, and so bring an additional leadership dynamic into the repo. Project leaders and core developers have a major impact on a repo. There are factors that affect a developers' chance to become repo leaders and/or core developers such as project environment and subjective willingness (Cheng et al., 2017).

Active GitHub developers submit repo commits, which improve software quality (Li et al., 2017). Another feature that GitHub offers is that of a reviewer/tester. They are high-quality assurance assets that provide developmental evaluation – usually under some minimum response timeframe (Li et al., 2017).

Yu et al. (2014b) suggest GitHub should engage a reviewer recommendation system, so appropriate reviewers/testers can be best-linked to each relevant incoming pull-request. Yu et al. (2014b) adds that social networks combined with information retrieval can deliver this system. The clarity of the source code, and its precision in

the documentation, encourage greater commit activity into the repo, and small documentation improvements can deliver great benefits (Henderson, 2009).

GitHub popular repos typically engage forking, they also show clearer, more consistent documentation advice (Aggarwal et al., 2014), and useful documentation can draw in other coding contributors (Hata et al., 2015). Such documentation may also be supported by testing mechanisms (Weber & Luo, 2014), Wikis (Hata et al., 2015), Twitter (Singer et al. 2014), social media and websites (Jiang et al., 2017).

When deciding whether to contribute to a GitHub repo, OSS developers often investigate a repo's popularity. This provides OSS developers with a calibration measure around the repo's success. The popularity of a repo is done by interpreting GitHub statistics in different ways (Xavier et al., 2014). Popularity is gauged by (Aggarwal et al., 2014; Xavier et al., 2014; Borges et al., 2015; Borges et al., 2016; Ma et al., 2016) against number of stars, forks, pull-requests and watchers.

In addition, popularity also relates to a repo's activity level (Cosentino et al., 2017). Other GitHub studies gauge various aspects of repo activity levels (Capra et al., 2011; Mileva, 2012; Bissyandé et al., 2013b; Weber & Luo, 2014; Zhu et al., 2014; Borges et al., 2016b). Each approach first adopts some form of clustering, possibly including programming language, duration, size, and social connections. This clustering allows each resultant dataset to be studied within a chosen modeling and/or coding and/or mathematical approach.

GitHub offers a range of components that assist in judging an OSS repo's activity levels (Härdle & Borke, 2017). Key GitHub programming languages are either web-focused (JavaScript, Ruby, PHP, CSS) or system-oriented (C, C++, Python). JavaScript, Java, and Python are currently the top three GitHub programming languages (Cosentino et al., 2017). From the above review, this thesis therefore selects the following aspects of GitHub repos on which to focus:

Repo-type: GitHub repos range from major corporate software developments such as Adobe bracket, or Facebook that incorporate forks when overcoming issues and/or when speeding new release versions, through to small core creator / developer repos.

Repo-lifetime: Large GitHub repos tend to remain active, forked, retain interest and be long-term ongoing operations (Cosentino et al., 2017). This thesis concentrates on mature repos where they were in GitHub for more than one year and they still gain more popular.

Repo-measures: GitHub measures commits, committers, software-releases, popularity-of-repo, number-of-stars-provided, forks, watchers, followers, testers, and reviewers (Xavier et al., 2014) (Härdle & Borke, 2017). In our dataset, the most forked and stars are the main criteria for the studied repos (see Chapter 3).

Repo-language: Key common GitHub software languages (discussed above) draw like-skilled developers and are more likely to retain repo communities in excess of 40 developers (Cosentino et al., 2017). Repo languages and variations are considered carefully in the selected dataset of this thesis (see Chapter 3).

Previous studies do not provide a holistic view of the constructs affecting a repo's activity level within GitHub repo ecosystems. Although the number of issues is also a repo success indicator some active repos do not engage GitHub's issue tracker (Cosentino & Cabot, 2015).

2.3.3 Thesis links to the literature

Table 2-1 summarizes the literature and shows the relationships between various GitHub measurable elements.

Table 2-1 Summary of previous studies regarding this thesis.

References	GitHub Elements	Strength of relationship	Investigate in this thesis
Hu et al., 2016; Borges et al. 2016a	Stars & Fork	Strong	✓
Borges et al. 2016a	Stars & Commits	Moderate	✓
Borges, et al., 2016b	Star & Contributors	Moderate	✓
Borges et al., 2016b	Release & Stars	Strong	✓
Kalliamvakou et al., 2016	Pull & Contributors	Strong	✓
Jiang et al. 2017; Vasilescu et al., 2015	Fork & Contributions	Strong	✓
Kalliamvakou et al., 2014	Commits & Pulls	Strong	✓
Peterson, 2013	Watchers & Fork	Strong	✓
Sheoran et al., 2014	Watchers & Commits	Negligible	✓

Forks, stars and contributors appear to be drives of existing literature, commits, pulls and watchers are relevant research foci. This thesis focuses on all of them and attempts to more deeply understand the relationships between them.

2.3.4 Mining GitHub and Challenges

GitHub is known as the 'absolutely dominant' data source for OSS data mining research (Cosentino et al., 2017). GitHub combines traditional capabilities including free hosting and version control with social features (Squire, 2014). Moreover, GitHub

supports rapid software development, and has collaborative repo features including bug-tracking, feature-requests, task-management and Wikis (Marlow et al., 2013; Zakiah & Fauzan, 2016). The interpretation of repository statistics is the subject of ongoing research (Borges et al., 2016; Cosentino & Cabot, 2016). The ability to understand GitHub repository statistics would allow developers easier access to repositories appropriate for consumption and allow developers to find repositories to which they would make suitable contributors. Kalliamvakou et al. (2016) suggest data sourced through mining GitHub is useful in evaluating aspects of software engineering provided those researchers datamining GitHub remain aware of what information they are pursuing.

Many studies have datamined GitHub to find user profiles, interpret customer behaviour and find repository preferences, such as programming language, popularity and usage (Ye & Kishida, 2003; Williams, 2012; Marlow et al., 2013; Gousios et al., 2014; Wu et al., 2014; Kalliamvakou et al., 2014; Blincoe et al., 2016). Cosentino et al. (2016) suggest that extracting knowledge by mining GitHub can be optimized for committers and/or repo collaborators.

Methods of collecting useful data and the size of available data are key concerns when mining GitHub (Kalliamvakou et al., 2016). Matragkas et al. (2014) use GHTorrent dataset to performed cluster analysis. This analysis showed that repo growth did not change the number of active repo contributors and/or the core team of repo developers. Moreover, the researchers found that passive users were the majority of members of large repos (Blincoe et al., 2016).

Although there are benefits to mining GitHub, many shortcomings remain (Gandomani et al., 2013). Studies to date lack: (1) longitudinal research, (2) predefined sampling techniques (for data extraction and analysis), (3) assessment using large data sets, (4) long data collection times needed to extract, collate, and deploy large data sets (Blincoe et al., 2016; Borges et al., 2016; Xavier et al., 2014), (5) software engineering team diversity and (6) software productivity consistency (Squire, 2017). Hence, long-term studies with clear datamining techniques that capture large datasets remain a knowledge gap in GitHub repo studies (Borges et al., 2016).

To improve pull request acceptance, Yu et al. (2016) applied a classifier evaluation matrix utilizing precision, recall and an F-measures. There are many aspects about GitHub repo research that represent threats to analysis and validity (Ray et al., 2014; Borges et al., 2016). Cosentino et al. (2017) suggest that a systematic study is needed, and study replication is lacking. Moreover, researchers indicate that GitHub API restrictions are a considerable challenge when trying to extract useful data about GitHub repos (Hebig et al., 2016).

Topic modeling is used to classify GitHub repos, which in turn is used to make recommendation system. Topic modeling also facilitates understanding about developer communities within GitHub and trends in GitHub software development (Bavota et al., 2014; Soll & Vosgerau, 2017). Orii (2012) applied a combination of topic modelling and collaborative filtering on GitHub repos, claiming that his method produces highly interpretable structures, but did not outperform existing methods. Markovtsev & Kant, (2017) applied topic modelling using repo name, however the

problem faced was the occurrence of duplicate repositories. There is a lack of studies about GitHub ecosystem (Xavier et al., 2014; Blincoe et al., 2016; Borges et al., 2016; Ma et al., 2017).

In conclusion, GitHub appears to be "the engine" of open source software development. GitHub's growing community of developers contribute from anywhere around the world at any time. To understand GitHub, researchers must be prepared to more deeply explore its nature. I believe that GitHub is an ideal place to explore OSS Ecosystem.

This research focuses on delivering a systematic model highlighting how the elements of the GitHub ecosystem relate together. It does not intend to test theory involving set elements within the GitHub ecosystem, and so does not require proof or mathematical modelling beyond SEM.

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Introduction

This Chapter describes the methodology, study design, sample data used, procedure and criteria used for data collection along with the mathematical tools used. Finally, the detail of the methods and techniques used for data analysis are covered.

3.2 Thesis Dataset

Data about GitHub repos are collected by using GitHub search tools. GitHub Querying is applied to collect an initial dataset. GitHub is currently the 'absolute dominant' data source for open source software (OSS) data mining research (Kalliamvakou et al., 2014; Hata et al., 2015; Dias et al., 2016; Cosentino et al., 2017).

GitHub's search engine allows searching of its repos against 'stars' or 'forks' counts (Jarczyk et al., 2014; Robinson & Deng, 2015; Borges et al., 2016b). The thesis considers the forks number as the main criteria to query GitHub for two reasons. Firstly, for a repo the presence of forks, means that this repo is active, and the open source software being developed is likely still under development. In contrast, the presence of stars just means that individuals express a likeness for the repo, and they've rated the repo with stars (Baudry & Monperrus, 2012; Weber & Luo, 2014).

Hence, the presence of forks offers a much stronger measure of the repo being active than does the presence of stars. Also, a repo user who forks the repo, likely has a high probability of generating a pull request which in-turn may affect (or increase) the

repo's net contributors number (Kalliamvakou et al., 2016). Secondly, most repos which acquire a high number of forks, also likely have a high number of stars (Borges et al., 2016a). Hence, this pilot study focusses on both GitHub search criteria.

Top three GitHub programming languages are JavaScript, Python and Java (Christopher et al., 2015; Cosentino et al., 2017a; Hu et al., 2016; Ray et al., 2015). These programming languages are used in both phases.

An API GitHub query collects eight datasets for case study two. It consists of the top eight programming languages JavaScript, Python, Java, C#, CSS, C++, Ruby and PHP (Badashian & Stroulia, 2016; Borges et al.,2016a; Kumar & Dahiya, 2017; Härdle & Borke, 2017; Noone & Mooney, 2017). Data set used in phase 1 are time series data representing 30 repos collected over six different time frames, two weeks between each time frame.

Each dataset in case study two captures eight GitHub key element which are: Stars, Forks, Watchers, Contributors, Releases, Issues (open or closed), Pulls (open or closed) and Commits. 1600 GitHub repos are downloaded, and prepared for analysis, each programming language data set consisting of its top 200 repos.

GitHub contains over 10 million repos (Kalliamvakou et al., 2014). Hence, querying the GitHub repos is an important step in understanding the data contained in various programming languages. Further, confirming the quality of an extracted sample of data also plays important role in the successful understanding of the data itself and in

the studies that draw on the data itself (Gousios & Spinellis, 2017). Borges et al. suggest extracting good sample data requires some degree of human interactions and the following of procedural guideline (Borges et al., 2016b; Cheng et al.,2018). In GitHub the majority of its repos are either inactive, or are personal (Kalliamvakou et al., 2014). In this thesis the analysis of GitHub's repos follows two main querying considerations: (1) the programming language and (2) the forks count.

The forks is an important measure of repo activity (Biazzini & Baudry, 2014; Chen et al., 2014 ; Jiang et al., 2017). The limitation conditions (or filters) applied for the repo's extraction applied to case study one is:

- Repos with more than one year old
- Illegal repos considered invalid and excluded from the dataset, this could be confirmed by visiting repos webpage.

These two limitation conditions applied to all this thesis's case studies. They ensured the consistency of the sample datasets, and upon look-up check all are dataset believed to represent good quality repos.

3.3 MATLAB program - developed for Querying GitHub

MATLAB program has been developed to extract GitHub sample data. MATLAB is reliable when dealing with URLs(Carpenter et al., 2017). The GitHub tool box located within MATLAB makes GitHub more flexible when transfer data between the two environments (GitHub and MATLAB). MATLAB also provides many evaluations and comparison instructions tools that rapidly offer assessments across large dataset (Higham & Higham, 2016; Pianosi et al., 2015). Thus, MATLAB helps in evaluating

the condition of repos. For example, MATLAB can compare the number of commits - considering the number of committers to a repo and eliminate questionable ones. In another word MATLAB tests if the GitHub repo's unique contribution condition for a commit is satisfied before extracting the commit element from the repo. Then MATLAB program extracts the commits from each repo, and directly adds the results into excel applications.

This process is used for all dataset collections and collations. The input into the MATLAB program is a useful CSV file. It is pair-labelled as 'Repo-Owner', 'Repo-Name'. This information is used to check and invoke repo attributes.

3.4 Study Design

This thesis addresses the research questions (pp. 10 Chapter one), and the supporting hypotheses regarding the GitHub ecosystem and its relationship with its key elements. These key elements exert an influence on an individual GitHub repo's success. Stepwise phases across these studies are illustrated in Figure 3-1. The methodology involves two overarching phases:

Phase one extracts GitHub data for a pilot study (3x10 GitHub repos). It is a short-term time series study looking at rate-of-change of the collected time series data and the competitive analysis statistical significance within and between top GitHub language differences.

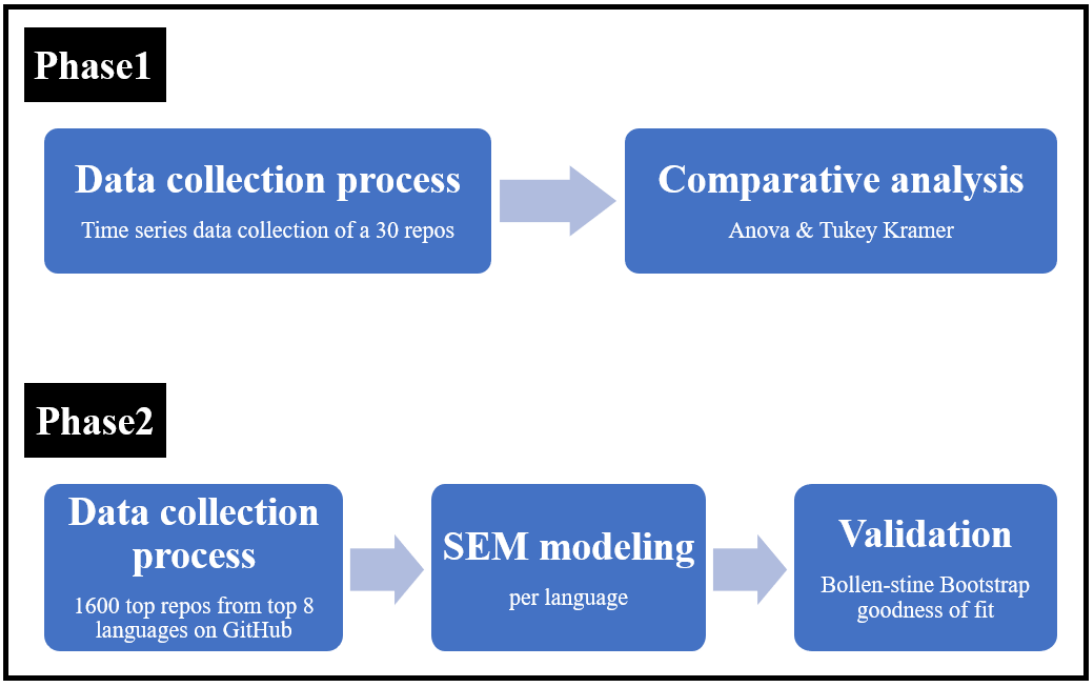


Figure 3-1: The two-phase methodology.

Phase two extract GitHub data for a large-scale modelling study involving 1600 of the most popular GitHub repos. In this thesis structural contribution models are delivered. Each model is bootstrap validated (200 times). An excellent bootstrapped model fit is validated when the Bollen-Stine p-value exceeds 0.05 (Hair et al.,1998).

3.5 Phase one- case study one

Phase 1 present case study one which is a pilot study, this phase consists of collecting snapshot data from GitHub to explore the existence of a difference in programming languages used in GitHub, Figure 3-2 shows main processes run across phase 1.

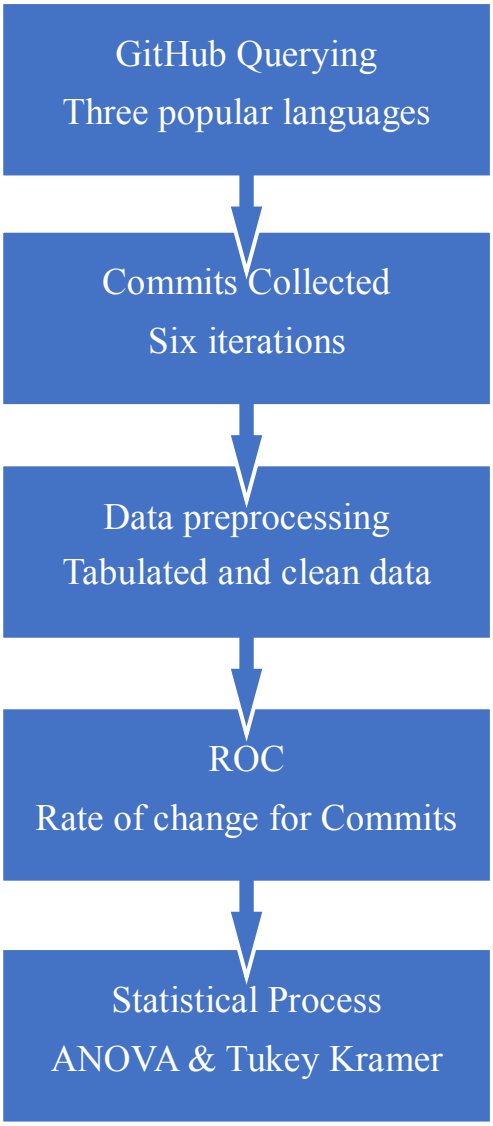


Figure 3-2: Phase 1 case study one processes.

3.5.1 Phase one- GitHub data collection and process

Querying GitHub using a 'general search' tool is the first step in data processing. The first filter is a programming language. Inside the programming language repos, only 'forked' repos are selected. The MATLAB program (now termed MATLAB) is used to access the repo's commit information. In case of study one (pilot study), three programming languages are used to search for language correlations. Each language

data captures only captures its own repos, and only uses the ten repos with the largest number of forks. These repos are collected firstly in 'pair-labelled' groupings – with the format: repo (owner, name) MATLAB uses to generate CSV files. Following a two weeks lag-time, the prior process is again repeated for same repos.

Case study one (the pilot Study), commits are considered, as according to the literature (Kalliamvakou et al., 2014) the activity in GitHub is mostly reflected by the level of commits. Commit is one of GitHub key elements. Metadata extractions for this pilot study occurred six times, two weeks apart, over a three months period. These evenly-spaced time intervals provided snapshots of the repositories elements measures per language.

Metadata for case study one is illustrated in Table 3-1. Which shows the repo name/owner as paired title and type- of- repos- representing the category of it used for. Each language specific data collection is allocated to a dedicated computer, and each involves the ten most popular repos per language.

Also, each selected repo must have been active for over three years – as this indicates the repo likely represents a substantive OSS program being developed (Vasilescu et al., 2016). Data extraction is performed using a custom tool[1] based on Version 3 of the GitHub API[2]. Data collection is then organized using the GitHub API group terms

[1] https://GitHub.com/ozyjay/GitHubQuery
[2] https://api.GitHub.com

(Chatziasimidis & Stamelos, 2015). This data is then tabulated, data cleaned, and checked for outliers (e.g. GitHub questionable repos[3]) which are removed.

This two week repetitive approach follows agile software development methodology (Gunal, 2012), and rapidly assesses the repositories development. This approach allows for changes or updates to each repo's information. Accordingly, repos that follow agile development methodology do show different activity levels at each collection point on the time-scale.

The resultant datasets total 180 (60 sets of data for each language) tabulated for comparative analysis. Initial findings indicated differences in repo activity levels between each language. Case study one investigates whether observed differences in GitHub repo occurred over time across these three programming languages.

[3] https://GitHub.com/shadowsocks/shadowsocks

Table 3-1: The top ten GitHub repos for JavaScript, Java, and Python

	JavaScript		Java		Python	
	Repo Name	**Repo Type**	**Repo Name**	**Repo Type**	**Repo Name**	**Repo Type**
1	twbs/bootstrap	Developer support	spring-projects/spring-boot	Code workaround	django/django	Developer support
2	angular/angular.js	Application software	spring-projects/spring-framework	Developer support	scikit-learn/scikit-learn	Coding style guide
2	udacity/frontend-nanodegree-resume	Coding style guide	alibaba/dubbo	Developer support	tensorflow/models	Coding style guide
4	d3/d3	Software library	elastic/elasticsearch	Developer support	pallets/flask	Developer support
5	facebook/react	Software library	iluwatar/Java-design-patterns	Coding style guide	ansible/ansible	Code workaround
6	jquery/jquery	Software library	zxing/zxing	Software library	udacity/fullstack-nanodegree-vm	Coding style guide
7	mrdoob/three.js	Software library	nostra13/Android-Universal-Image-Loader	Software library	vinta/awesome-Python	Developer support
8	freeCodeCamp/freeCodeCamp	Coding style guide	aporter/coursera-android	Software library	fchollet/keras	Software library
9	facebook/react-native	Coding style guide	jfeinstein10/SlidingMenu	Software library	odoo/odoo	Application software
10	tastejs/todomvc	Developer support	square/okhttp	Application software	josephmisiti/awesome-machine-learning	Software package

Repo collection procedure was re-executed every two weeks for each programming language, and for the same repos. Thirty repo commits are collected in timeframe 1, another 30 commits for the next timeframe till timeframe 6. Total number of repos where 180 repos (3 languages x 10 repos x 6 samples). Collecting six consecutive timeframes allows for a suitable trend analysis to be established. Moreover, agile software development practices suggest that a major coding milestone typically takes at least 6 iterations to produce a stable release of the software where each iteration typically takes two weeks.

3.5.1.1 Phase one- Rate of Change

Analysis of the data collected in case study one applies Rate of Change (ROC) as a normalization process (Campos & Scherson, 2000) as shown in Figure 3-2. This process makes the data easier to deal with. Let ROC be defined as a normalized value that measures how a quantity changes over a fixed time interval.

$$ROC = 100 \left[\left(\frac{current\ quantity}{previous\ quantity\ n\ time\ periods\ ago}\right) - 1\right]$$

3.5.1.2 Phase one- Comparative analysis

Next ANOVA (analysis of variance) is applied to investigate where significant differences arise (Anderson, 2001). Finally, the Tukey-Kramer method is used to find any differences between the three GitHub programming languages (Cho, 2014).

3.5.1.3 Statistical tools

To compare the three most popular GitHub programming languages a standard one-way ANOVA (analysis of variances) and Tukey-Kramer (post hoc) is deployed to confirm the validity of sample data used in the pilot study (King, 1986), and to discover if any significant differences between programming languages arise over time (Hair, et al. 1998).

One-way analysis of variance (ANOVA) tests the equality of three or more means at one time by using variances (Melosan, 2014). ANOVA determines whether any such variation is the result of some factor, or is simply the result of randomness, and ANOVA assumes:

- each comparison population is normally distributed;
- the observations are independent of one another; and
- each of the comparison populations displays an equivalent variance.

The Tukey-Kramer method (TK) is widely deployed in multiple comparison procedures (Benjamini & Braun, 2002) (Driscoll, 1996). TK considers possible pairwise differences of means at the same time and identifies pairs of means showing significant differences. In this thesis TK locates the differences within each programming languages (the difference between repos of the same programming languages- “difference within”) and the differences among different programming languages “difference between” TK assumes:

- observations being tested are independent within and among the language groups;
- language groups associated with each mean are normally distributed; and

- homogeneity of variance.

Results using these tools are discussed in Chapter four.

3.6 Phase Two -Case study two

Top good GitHub repos are collected, with each repo having more than two contributors (Borges et al., 2016b; Kalliamvakou et al., 2014). According to Kalliamvakou et. al. (2014) good repos also show a balance between pull request and commits. Figure 3-3 illustrates the data extraction steps used in case study two.

Case study two deploys eight top programming languages : JavaScript, Python, Java, C#, CSS, C++, Ruby and PHP (Onoue et al., 2013; Borges et al., 2016a; Badashian & Stroulia, 2016; Kumar & Dahiya, 2017; Härdle & Borke, 2017 ; Noone & Mooney, 2017). It uses larger data about 1600 repos (200 repos for each language).

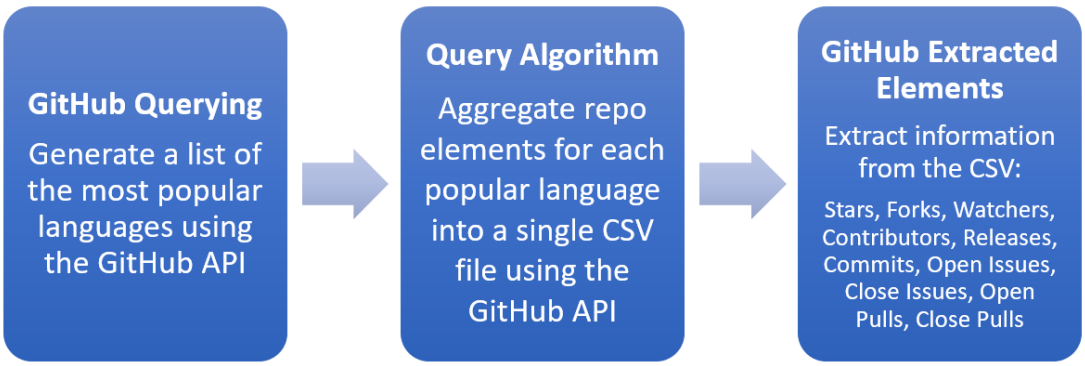


Figure 3-3: General Steps in Data Collection Process.

3.6.1 Case study two-Data collection

The first step in data collection is to query the GitHub repos – applying GitHub repo condition (filter). The limitation conditions (or filters) applied for the repo extraction applied to case study two are:

- Repos with an unbalanced number of commits and committers are excluded from the sample(Barnett et al., 2016; Goyal et al., 2018);
- Repos engaged each possessed more than two contributors to be considered valid; and
- Education repos are excluded from the sample (such repos have much forks number but commits number not changing).

In this thesis each repo captures the ten GitHub key elements: Stars, Forks, Watchers, Contributors, Releases, Issues (open & close), Pulls(Open & close) and Commits (Kalliamvakou et al., 2016). These ten variables represent the key open source software development contributing groups that collaboratively help build a GitHub repo over time. Table 3-1 shows and defines these ten GitHub key elements.

Table 3-2: GitHub repository data attributes and associated meanings.

Element Name	***Type/Classification***	***Meaning***
Stars	Repo interest	Developers who like the repo
Watchers	Repo interest	Developers who get a notification when the repo content changes
Forks	Repo interest	Isolated versions of a repo where changes are made to the original content or its intent
Commits	Repo work	Content changes resulting from new features, refactoring, and incremental development
Contributors	Repo work	Developers who asked to directly contribute to a repo
Releases	Repo work	Milestones in the lifetime of a repo
Issues open	Change request	Identified problems with repo content
Issues closed	Change request	Issues fixed by commits or merges after a review process
Pulls open	Change request	Suggested commits from forked versions of the repo or within the repo from developmental branches
Pulls closed	Change request	Pulls merged into the repo after a review process

Both stars and forks ratings are used for rank GitHub Repos, and both display a strong correlation (Vasilescu et al., 2015; Hu et al., 2016; Borges et al., 2016). Watchers are a good measure of how good repo is, and they too have a similar rating influence to stars (Badashian & Stroulia, 2016; Borges et al., 2016b).

The forks is a first step in making contributions (Jiang et al., 2017). The forks of a repo mean making a local copy of that repo, developer may or may not choose to make update after looking at their own local copy. If developer make an update, then a pull request may be sent to either fix a bug or add a new feature or make a modification.

Issues help in assigning, managing resources and eliminating software failures (Liao, Dayu, et al., 2018). A pull request is important to measure, and allows opportunities

for engagement - allowing more developers into the repos' community (Kalliamvakou et al., 2016). Whenever a pull request is sent to the repo community's, it marked as open pull request.

The repo-project-creator/developer (or core development team) reviews any open pull request. They either accept the pull request which results in adding a new contributor, or they reject the pull request. Usually pull requests of a non-technical nature are often rejected. In either case (accepting or refusing) the pull request is closed(Padhye et al., 2014; Kalliamvakou et al., 2014) (Tsay et al., 2014). Releases also have an influence on repos, Borges et al found that the number of starts increases rapidly upon issuing a new releases (Borges et al., 2016a). Table 3-3 provides a snapshot of sample data resulted from querying and collecting GitHub repos for case study two. The Table below shows all ten key elements for each repo display significant activity levels (and suitable for statistical analysis).

Table 3-3: The data sample of case study two: Top 20 Repos of Python language

REPOS(Owner-Name/Repo-Name)	Watch	Star	Forks	Commits	Releases	Contributors	Issue open	issue closed	Pull Open	Pull Closed
tensorflow/models	2108	31221	17290	1982	3	327	506	1721	197	1195
scikit-learn/scikit-learn	2062	26763	13468	22663	86	1040	994	3865	598	5395
ansible/ansible	1866	29100	10571	36383	201	3341	3463	13355	1411	19530
pallets/flask	1979	34082	10483	3202	21	458	23	1378	2	1257
keras-team/keras	1595	27258	9950	4423	40	644	1187	5743	30	2754
udacity/fullstack-nanodegree-vm	26	198	9271	53	0	6	13	11	12	51
vinta/awesome-Python	4088	47336	9139	1220	0	283	44	57	277	659
odoo/odoo	1311	9109	7809	115419	81	760	1162	6870	816	13257
josephmisiti/awesome-machine-learning	2816	31425	7693	1029	0	304	6	39	0	441
scrapy/scrapy	1666	26311	6590	6614	81	281	375	1147	197	1457
XX-net/XX-Net	1736	21625	6449	1824	247	60	7143	2363	2	419
rg3/youtube-dl	1411	35140	6443	16016	976	599	1878	11243	280	2468
requests/requests	1245	31266	5757	5416	131	497	95	2472	16	1736
pandas-dev/pandas	812	13551	5501	16911	89	1115	2328	9669	159	8284
apache/incubator-mxnet	1133	13432	4951	6780	42	495	752	4849	63	4499
wangshub/wechat_jump_game	557	13594	4869	324	2	65	26	949	0	253
tornadoweb/tornado	1047	15390	4485	3701	50	280	130	1116	57	1013
saltstack/salt	598	8692	4058	92042	163	2026	3567	14310	112	28663

Table 3-3 presents a selection of the 1600 repos investigated in case study two – 200 from each of the eight-top programming languages are collated, dataset organized and data-cleaned. Any questionable repo such as shadowsocks/shadowsocks is then removed before analysis. Each cleaned dataset is used to separately explore how the elements within an individual programming language may relationally fit into a language-specific structural path model.

Comparison of the eight path models then indicates whether path model differences exist between each of the three programming languages.

3.6.2 Case study two - Structural Equation Model

Case study two draws upon three theories (information integration, planned behaviour, and social translucence) to help frame this thesis's structural path model approach. These theories help establish a framework through which to capture the ten GitHub key elements.

The structural path model approach identifies the significant paths and relative path strengths between elements (termed constructs in path modelling). However, such data collection assumes measures are made without random measurement error. As this feature can disguise multicollinearity effects (Joseph F Hair et al., 1998; Grapentine, 2000), it is controlled by only engaging: (1) one GitHub software language at a time, (2) top (highly active) GitHub-specific repos, (3) currently active GitHub repos, and (4) GitHub repos with a continual repository longevity exceeding more than one years.

The GitHub programming language structural path model also offers a total effects Table which can elucidate an understanding of the total effects each elements element has across the programming language path model, and through to the outcome dependent variable of commits-per-language.

Eight structural path models each initially representing 200 repos for the eight-top programming languages (JavaScript, Python, Java, C#, CSS, C++, Ruby and PHP) are developed using AMOS 25.0. These eight path models, complete with their elements individual total effects Table, offer repo-project-creator/developer (or core development team) new understanding concerning how each GitHub contributor engages with some active GitHub repos.

The structural path model approach then allows a repo-project-creator/developer (or core development team) to pursue additional ways to possibly: (1) draw further OSS developers into this repo, (2) induce higher repo element activity levels, and (3) shorten the time between repo releases versions.

In the structural path model pulls open and pulls closed are considered as one ‘pulls’ activity - since every closed pull was formerly an open pull allowed by the repo. Issues open, and issues closed are also similarly combined as the repo’s ‘issues’. Thus, pulls are the summation of pulls open and pulls closed, and issues is the summation of issues open and issue closed.

The GitHub ten key elements represent different variables used in the structural model. The three types of variables are: independent, intermediate and dependent variables, each programming language will have processed using path analysis three out of ten GitHub elements represent independent variables these are forks, watchers and stars. Fork, stars and watchers are strongly correlated to each other (Peterson, 2013; Badashian & Stroulia, 2016; Borges et al., 2016b; Hu et al., 2016) each of these variables used to rank GitHub repos and have the same influence on GitHub repos rank. While commits represent dependent variables others are intermediate variables Figure 3-4 depicted these different variables name and types.

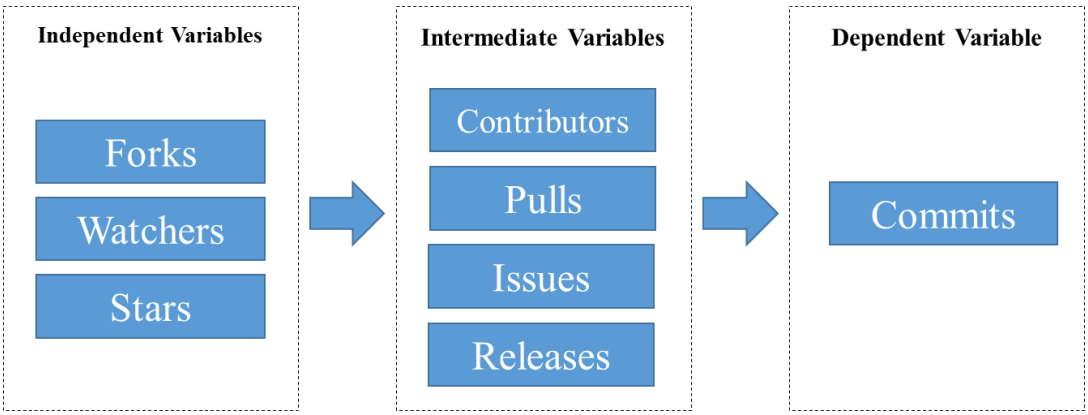


Figure 3-4: Variables name and types used in structural model.

Pulls, releases contributors and issues each one of these variables may affect by each other these four GitHub key elements represents intermediate variables (Padhye et al., 2014; Tsay et al., 2014; Xavier, 2015; Kalliamvakou et al., 2016). Pull request, releases forks all these variables will lead to increase commits, thus commits is the dependent variables (Kalliamvakou et al., 2014).

The AMOS 25.0 software offers model fit validations. Key fit excellence measures include: are: Chi-square ($\chi 2$), degrees of freedom (DF), p-value (or Bollen-Stine p-value), Root Mean Square Error of Approximation (RMSEA) and various model fit measures.

Chi-square remains a key measure. It less sensitive to sample size, when measured as $\chi 2$/DF, and ranging within 1 to 3, and showing a p-value above 0.05, it indicates the structural model is a very good model fit (Byrne, 1994; Hair et al. 1998, Ullman, 2006, Kline, 2015).

RMSEA estimates the average absolute difference between the path model's covariance estimates and its observed covariances. Values below 0.05 indicate an excellent fit, but RMSEA values below 0.08 also indicate very good to near excellent fit (Steiger, 1990, Browne & Cudeck, 1992; Hu & Bentler, 1998).

Many 'goodness of fit' measures exist. Comparative Fit Index (CFI) should be above 0.95, Tucker-Lewis Index (TLI) should be above 0.95, and Normative Fit Index (NFI) should be above 0.95 (Hu & Bentler, 1998). The Goodness-of-Fit Index (GFI) measures the fit between hypothesized model and should be above 0.90 (Byrne, 1994; Baumgartner & Homburg, 1996), and Absolute-Goodness-of-Fit Index adjusts the GFI with a degree of freedom inclusion, and should be above 0.90 (Hu & Bentler, 1998).

3.7 Phase two-Validation

For path model validation each programming language's path model is bootstrapped 200 times. Structural path model validations are undertaken via bootstrapping 200 times to capture and average item variations. Here a Bollen-Stine p-value of above 0.05 is desired. Chapter four will explore the obtained results from both Phases.

CHAPTER 4

RESULTS

4.1 Introduction

This Chapter presents the results of the exploratory GitHub pilot study. This Chapter deploys Excel, SPSS and AMOS programs to analyses each of the GitHub language data sets. Phase one engages GitHub's 30 most contributed repos – 10 from each of its three top programming languages (JavaScript, Python and Java). It tests the existence of significant differences among/within these GitHub programming languages. Phase two then builds the structural equation model (SEM) for each of the top eight programming languages. The significant SEM construct pathways that contribute towards delivering the GitHub language commits are shown. These indicate each language's repo activity levels. SEM model goodness-of-fit assessments are also provided.

4.2 Phase one

The pilot study engages 30 of the top GitHub repos - sorted by "most Forks." The ten most active repos for each programming language are utilized. These top programming languages are JavaScript, Python and Java (Cosentino et al., 2017; Härdle & Borke, 2017; Noone & Mooney, 2017).

Although GitHub has many key elements used by researchers, 'commits' is the measurement item that reflects GitHub's activity level and its productivity (Cortés-Coy et al., 2014; Kalliamvakou et al., 2014; Goyal et al., 2018).

Tables 4-1, 4-2 and 4-3 show this pilot study sequentially collected 'commits' across six time periods (Commit1 to Commit6) for each of three GitHub languages JavaScript, Java and Python. In all bar three cases, each GitHub repo's activity level grows and remain significant. In four cases there is no increase or decrease over the six time periods studied. This may be due to an OSS repo being in an on-hold phase with no activity growth needed or operating stably and requiring no change.

Table 4-1: JavaScript data set collected over six timeframes*.

Repo	Commit1	Commit2	Commit3	Commit4	Commit5	Commit6
1	16976	17096	17221	17268	17311	17380
2	8597	8610	8621	8625	8648	8655
3	84	84	84	84	84	84
4	4104	4104	4104	4104	4106	4110
5	9158	9269	9325	9398	9529	9547
6	6268	6270	6270	6271	6275	6276
7	20530	20654	20930	21066	21403	21573
8	10691	10706	10749	10756	10816	10876
9	11939	12065	12139	12219	12357	12439
10	2828	2829	2829	2832	2832	2832

Table 4-2: Java Commit data set collected over six timeframes*.

Repo	Commit1	Commit2	Commit3	Commit4	Commit5	Commit6
1	13665	13833	14064	14226	14540	14640
2	15461	15533	15615	15694	15773	15798
3	1921	1940	1967	1971	2000	2054
4	28879	28941	29034	29144	29360	29694
5	1854	1870	1879	1890	1931	1945
6	3375	3380	3399	3404	3407	3410
7	1025	1025	1025	1025	1025	1025
8	71	71	71	71	71	71
9	336	336	336	336	336	336
10	3060	3061	3061	3070	3073	3079

Table 4-3: Python Commit data set collected over six timeframes*.

Repo	Commit1	Commit2	Commit3	Commit4	Commit5	Commit6
1	25009	25064	25111	25152	25221	25262
2	22265	22315	22348	22385	22452	22479
3	1298	1318	1388	1458	1531	1569
4	3085	3094	3099	3101	3114	3126
5	33338	33587	33840	34088	34516	34732
6	47	47	47	48	48	48
7	1152	1154	1159	1161	1171	1173
8	3921	3972	4025	4089	4152	4174
9	113132	113471	113651	113866	114183	114242
10	914	928	946	962	981	989

* Note: the detailed description of the six timeframes is explained in Section 3.5.1.

4.2.1 Pilot study – Rate of Change (ROC)

Each Table 4-1,4-2 and 4-3 data set is normalized as a ROC to gauge the percentage increase or decrease in commits over a given period of time (Heirich, 1964). Tables 4-4,4-5 and 4-6 illustrate the normalizing (ROC) of JavaScript, Java and Python respectively. This ROC clarifies that for each time interval the ROC remains positive – indicating a growth in activity levels is occurring across the top ten GitHub repos for each language.

Table 4-4: JavaScript normalized data.

Commit/Repo	1	2	3	4	5	6	7	8	9	10	Average
Commit1	0.71	0.15	0.00	0.00	1.21	0.03	0.60	0.14	1.06	0.04	0.39
Commit2	0.73	0.13	0.00	0.00	0.60	0.00	1.34	0.40	0.61	0.00	0.38
Commit3	0.27	0.05	0.00	0.00	0.78	0.02	0.65	0.07	0.66	0.11	0.26
Commit4	0.25	0.27	0.00	0.05	1.39	0.06	1.60	0.56	1.13	0.00	0.53
Commit5	0.40	0.08	0.00	0.10	0.19	0.02	0.79	0.55	0.66	0.00	0.28

Table 4-5: Java normalized data.

Commit/Repo	1	2	3	4	5	6	7	8	9	10	Average
Commit1	1.23	0.47	0.99	0.21	0.86	0.15	0.00	0.00	0.00	0.03	0.39
Commit2	1.67	0.53	1.39	0.32	0.48	0.56	0.00	0.00	0.00	0.00	0.50
Commit3	1.15	0.51	0.20	0.38	0.59	0.15	0.00	0.00	0.00	0.29	0.33
Commit4	2.21	0.50	1.47	0.74	2.17	0.09	0.00	0.00	0.00	0.10	0.73
Commit5	0.69	0.16	2.70	1.14	0.73	0.09	0.00	0.00	0.00	0.20	0.57

Table 4-6: Python normalized data.

Commit /Repo	1	2	3	4	5	6	7	8	9	10	Average
Commit1	0.22	0.22	1.54	0.29	0.75	0.00	0.17	1.30	0.30	1.53	0.63
Commit2	0.19	0.15	5.31	0.16	0.75	0.00	0.43	1.33	0.16	1.94	1.04
Commit3	0.16	0.17	5.04	0.06	0.73	2.13	0.17	1.59	0.19	1.69	1.19
Commit4	0.27	0.30	5.01	0.42	1.26	0.00	0.86	1.54	0.28	1.98	1.19
Commit5	0.16	0.12	2.48	0.39	0.63	0.00	0.17	0.53	0.05	0.82	0.53

4.2.2 Pilot study – ANOVA

To test the hypothesis of the existence of a difference in each programming language and to assure the validity of the data, ANOVA test was applied for each programming language (ROC Tables 4-4, 4-5 and 4-6). The results of applying ANOVA test illustrated in tables 4-7, 4-8 and 4-9 for JavaScript, Python and Java programming languages respectively.

To test the hypothesis that each programming language displays differences, and to assure the validity of the data, the ANOVA test is applied for each programming language (ROC Tables 4-4, 4-5 and 4-6). These results, illustrated in Tables 4-7, 4-8 and 4-9 for JavaScript, Python and Java programming languages respectively suggest significant difference exists across groups in Table 4-7, Table 4-5, Table 4-6.

Table 4-7: Result of applying ANOVA to JavaScript normalized data.

ANOVA			**DV= Commits**			
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	6.86	9	0.76	12.47	0.00	2.12
Within Groups	2.44	40	0.06			
Total	9.30	49				
			Level of significance			0.05

Table 4-8: Result of applying ANOVA to Python normalized data.

ANOVA			**DV= Commits**			
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	59.37	9	6.6	14.58	0.0000	2.12
Within Groups	18.1	40	0.45			
Total	77.47	49				
			Level of significance			0.05

Table 4-9: Result of applying ANOVA to Java Language normalized data.

ANOVA			**DV= Commits**			
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	13.58	9	1.51	8.17	0.0000	2.12
Within Groups	7.39	40	0.18			
Total	20.98	49				
			Level of significance			0.05

4.2.3 Pilot study – Tukey-Kramer

To investigate differences between the three programming languages, this study deployed the Tukey-Kramer method. Table 4-10 summarizes the Tukey-Kramer results for the JavaScript, Python and Java programming language. Results indicate Python is different to JavaScript and Java, and Java is sometimes different to JavaScript. Table 4-10 summaries result of applying Tukey-Kramer between three programming languages difference.

Table 4-10: Tukey-Kramer results for three GitHub languages.

Comparison	Sample Mean	Sample Size	Absolute Difference	Std. Error of difference	Critical Range	Results
JavaScript vs Java	0.03	10	0.1	0.13	0.47	Means are not different
JavaScript vs Python	0.12	10	1.56	0.13	0.47	Means are different
Java vs Python	1.59	10	1.47	0.13	0.47	Means are different

4.3 Phase Two

The structural path model approach is regression based. It is applicable where models are not too complex (Grapentine, 2000). Within GitHub Repo ecosystem, structural path analysis and SEM modelling capture the key GitHub measurement constructs: Forks, Watch, Stars, Issues (open & closed), Releases, Pulls (open & closed), Contributors and Commits. For each language, the Standardized Total Effects of each of GitHub measurement construct then indicates each construct's net effects onto the GitHub language's repos activity level (as measured by its commits).

The top 1600 GitHub repos for JavaScript, Python, Java, , C#, C++,CSS, PHP, and Ruby programming languages are assessed across various pathways of repo contribution. Data captured from these popular differ in: format, repo-size, development-cycle-stage, change-frequency, change-degree, Forks, Watch, and contributor-skills, and the data, has remained difficult to interpret.

4.3.1 Path Analysis Model

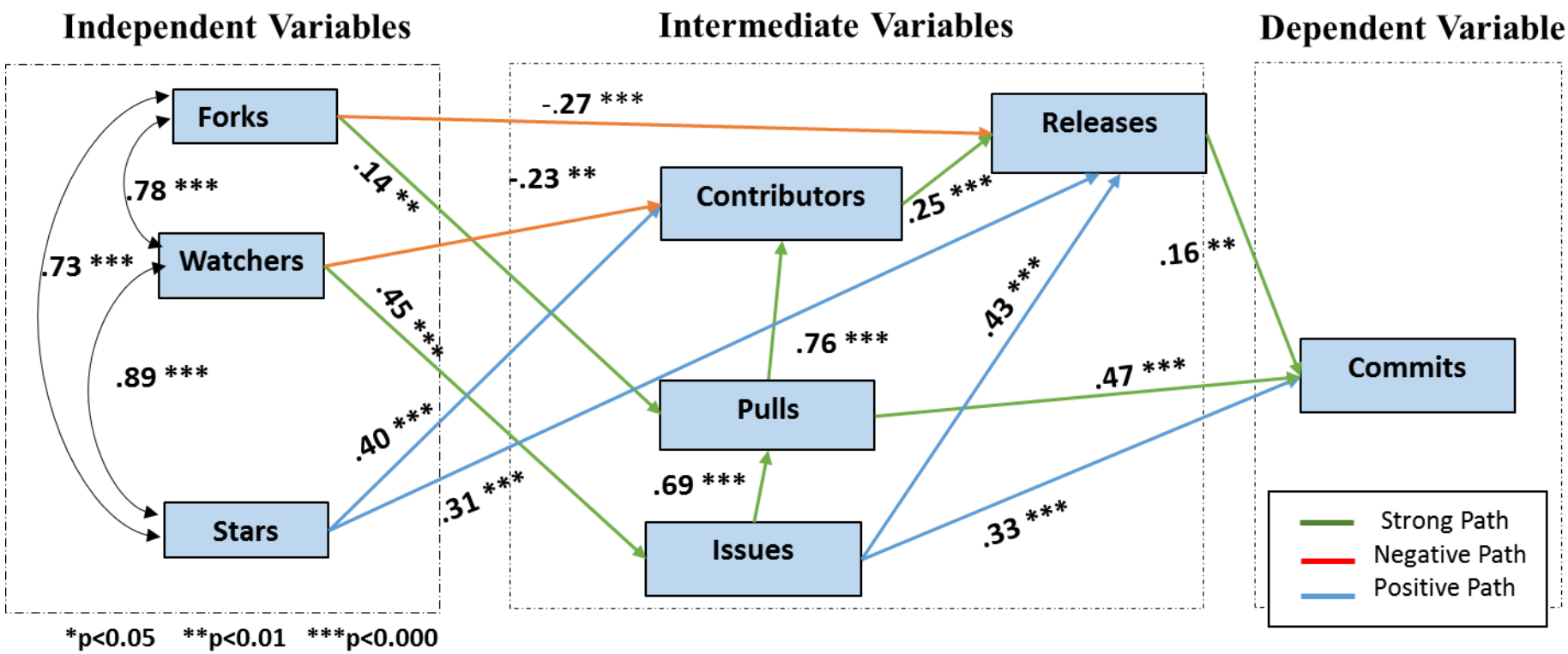
This GitHub structural path model study utilizes 1600 repos - 200 repos for each of the eight most-used (top) programming languages. These languages - JavaScript, Python,

Java, C#, C++,CSS, PHP, and Ruby (Jain & Gupta, 2017; Kumar & Dahiya, 2017) are collected if they are popular and have been in operation for more than one year. Each reop captures the key GitHub measurement constructs: Forks, Watch, Stars, Issues (open & closed), Releases, Pulls (open & closed), Contributors and Commits). The total dataset size-contributions of the key GitHub measurement constructs for the 1600 repos investigated is illustrated in Table 4-11. All languages show each of the key GitHub measurement constructs provide consistently high OSSD repo contributions. Considering JavaScript its 200 repos dataset has five outliers – which are discarded as they fail to satisfy the limitation condition (stated in Chapter three Section 3.2.3.4).

For the Python language again 5 outliers are removed, and an excellent fit SEM model result. Figure 4-2 represents Python language structural path model. Figure 4-3 represents the Java programming language. Here, 30 repos are outliers failing to satisfy the limitation condition (filters in Chapter 3). Hence, the modelled Java dataset uses 170 repos. Figures 4-4, 4-5, 4-6, 4-7 and 4-8 respectively represent the structural path models for the GitHub repos for C#, C++, CSS, PHP and Ruby programming languages.

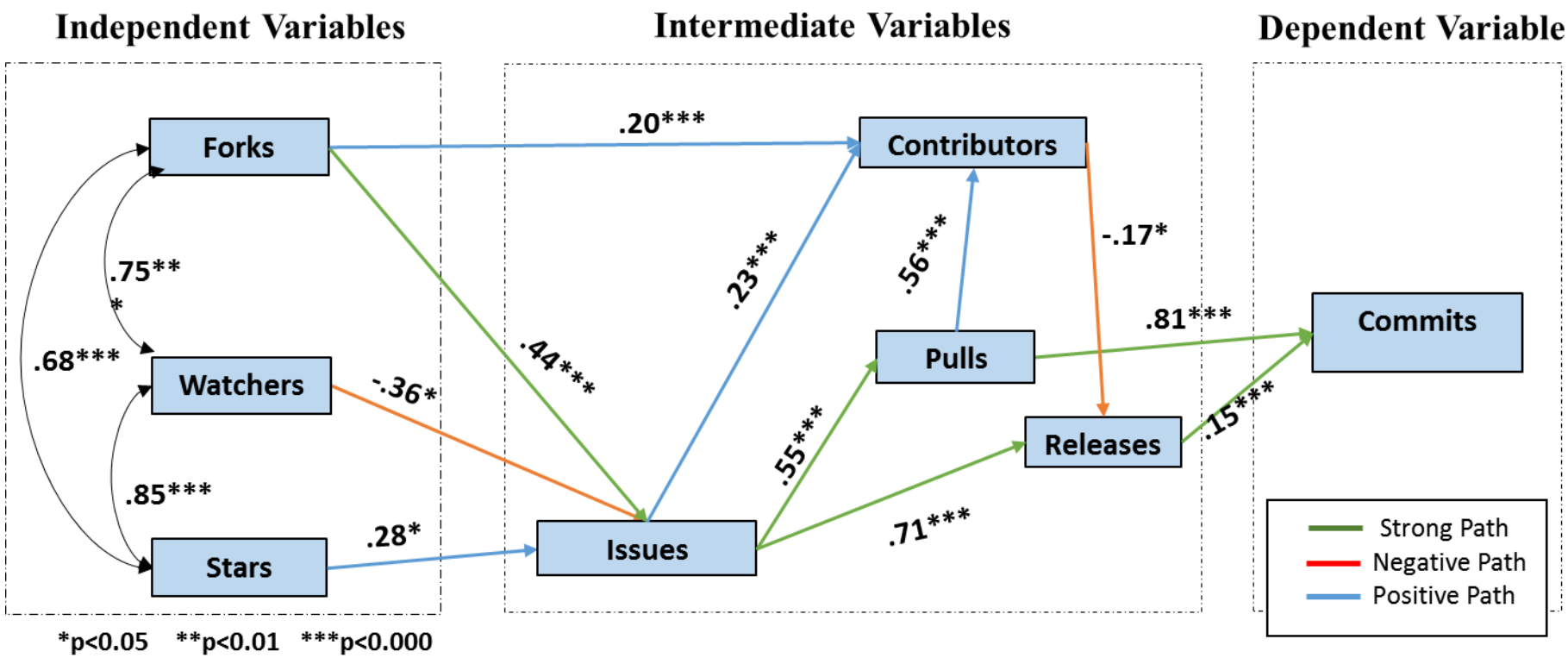
Table 4-11: GitHub dataset – the top 1600 repos for the top eight languages.

GitHub Elements	Programming Language							
	JavaScript	**Python**	**Java**	**C#**	**C++**	**CSS**	**PHP**	**Ruby**
Watches	165153	100346	119342	61243	93013	50,071	55612	43034
Stars	3553983	1551498	1495595	467021	1077757	954,704	740185	957489
Forks	809328	426286	527948	150931	365894	298,475	233805	293345
Releases	15324	10897	9230	10596	10295	2,806	13256	25021
Contributors	46707	44978	14563	14859	23826	12,325	32611	59756
Issue Open	52235	62522	33213	44004	57886	9,085	34774	18065
issue Closed	382345	233734	152421	182004	212136	50,930	269520	185834
Issues	434580	296256	185634	226008	270022	60,015	304294	203899
Pull Open	10792	11191	4122	4130	6489	2,133	5682	18199
Pull Closed	230585	295534	126353	151627	234728	41,036	240669	312968
Pulls	241377	306725	130475	155757	241217	43,169	246351	331167
Commits	708605	1178386	941579	659663	1395130	155,084	1106250	1777399



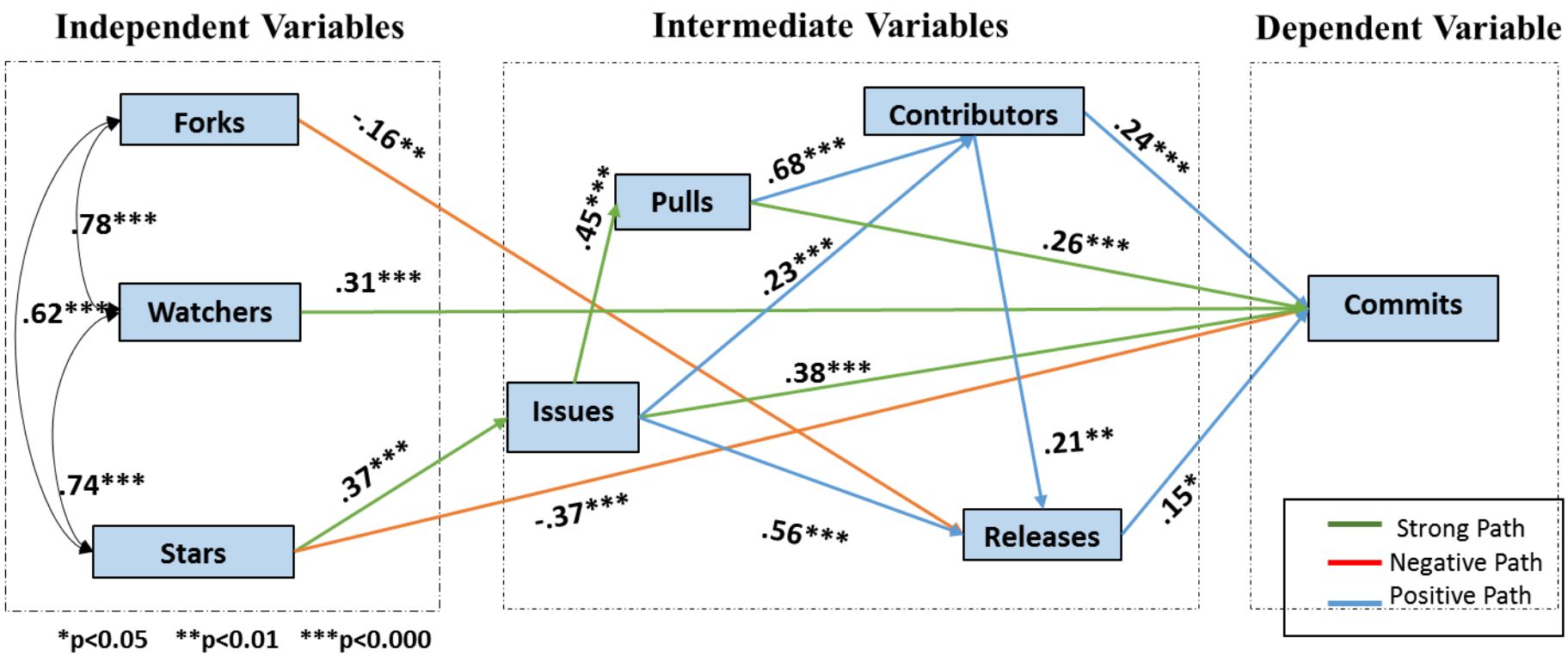
Relative Chi-Sq (X2/df)	Probability level (p)	RMSEA	TLI	CFI	GFI	AGFI
1.678	0.064	0.059	0.985	0.994	0.996	0.927

Figure 4-1: JavaScript programming language Path Model.



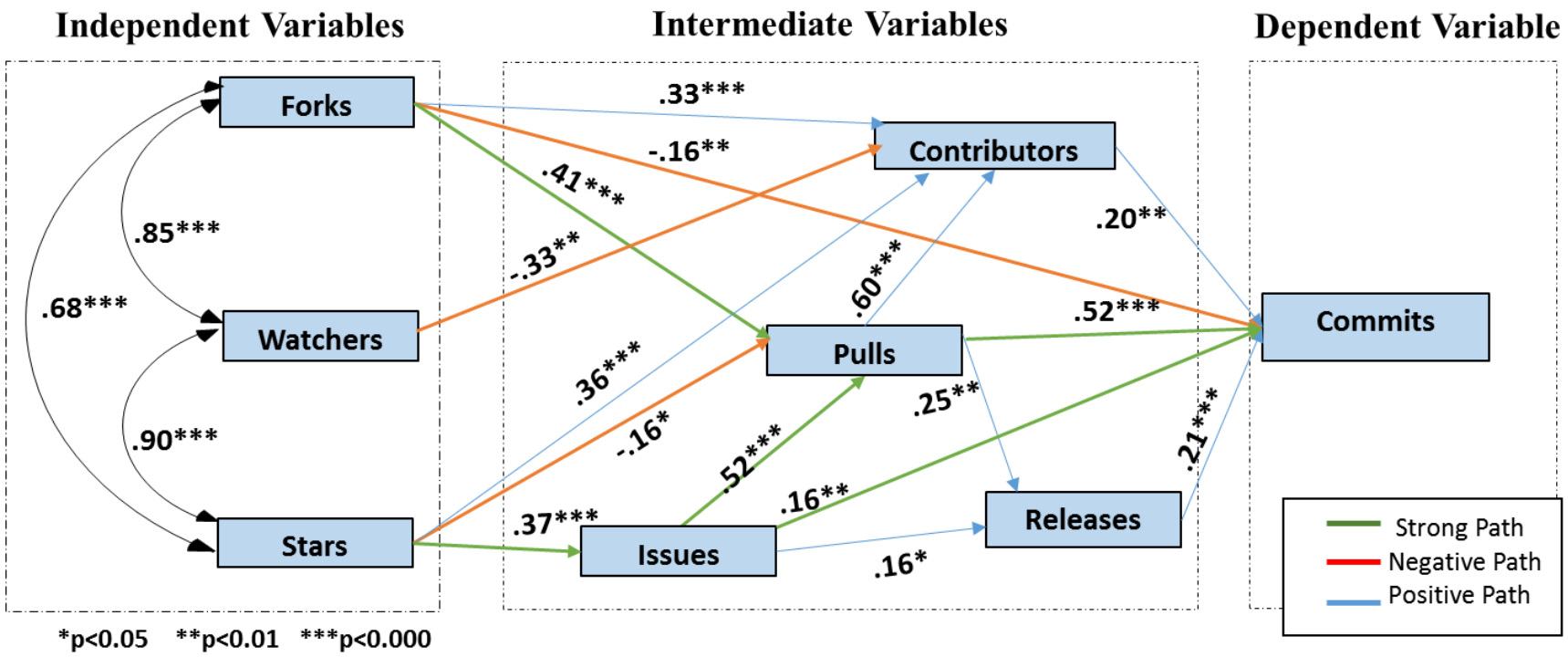
Relative Chi-Sq (X2/df)	Probability level (p)	RMSEA	TLI	CFI	GFI	AGFI
1.555	0.083	0.054	0.985	0.993	0.974	0.932

Figure 4-2: Python Programming Language Path Model.



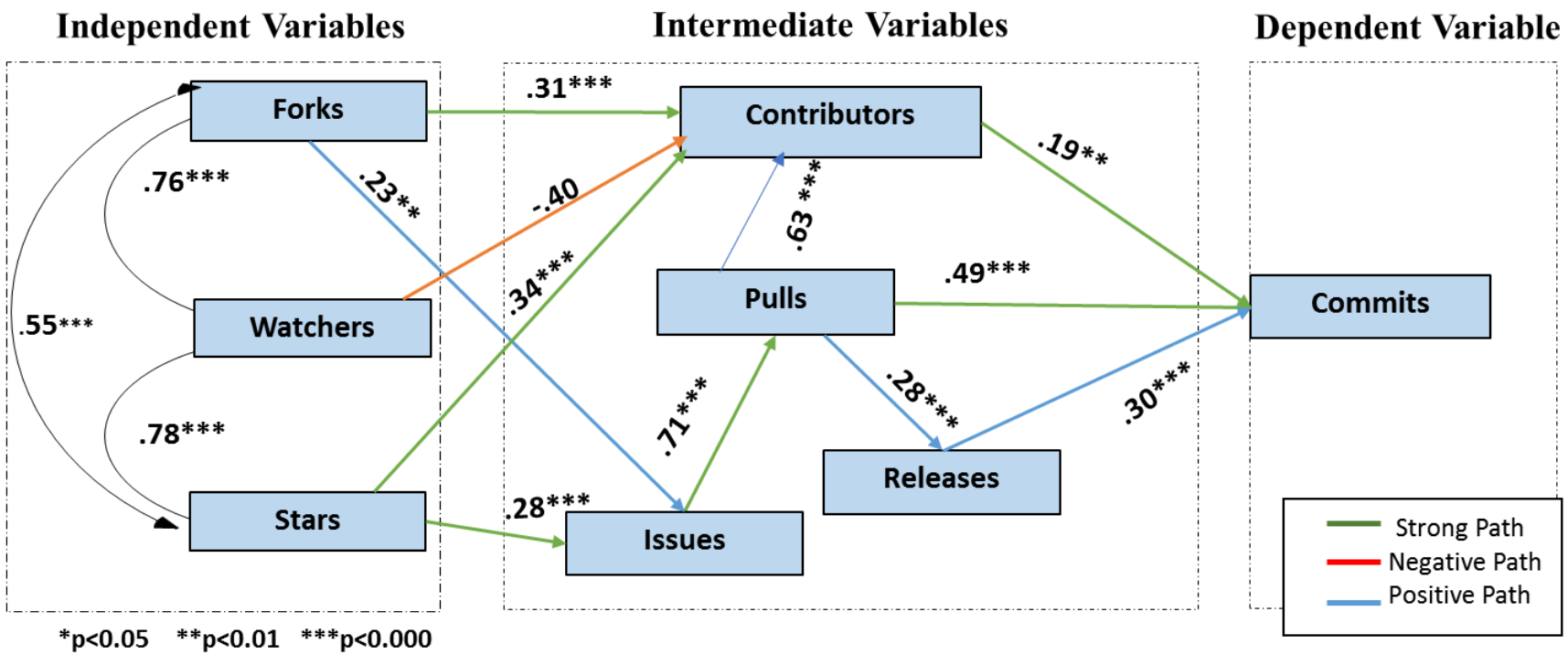
Relative Chi-Sq (X2/df)	Probability level (p)	RMSEA	TLI	CFI	GFI	AGFI
1.75	0.502	0.067	0.975	0.988	0.968	0.905

Figure 4-3: Java Programming Language Path Model.



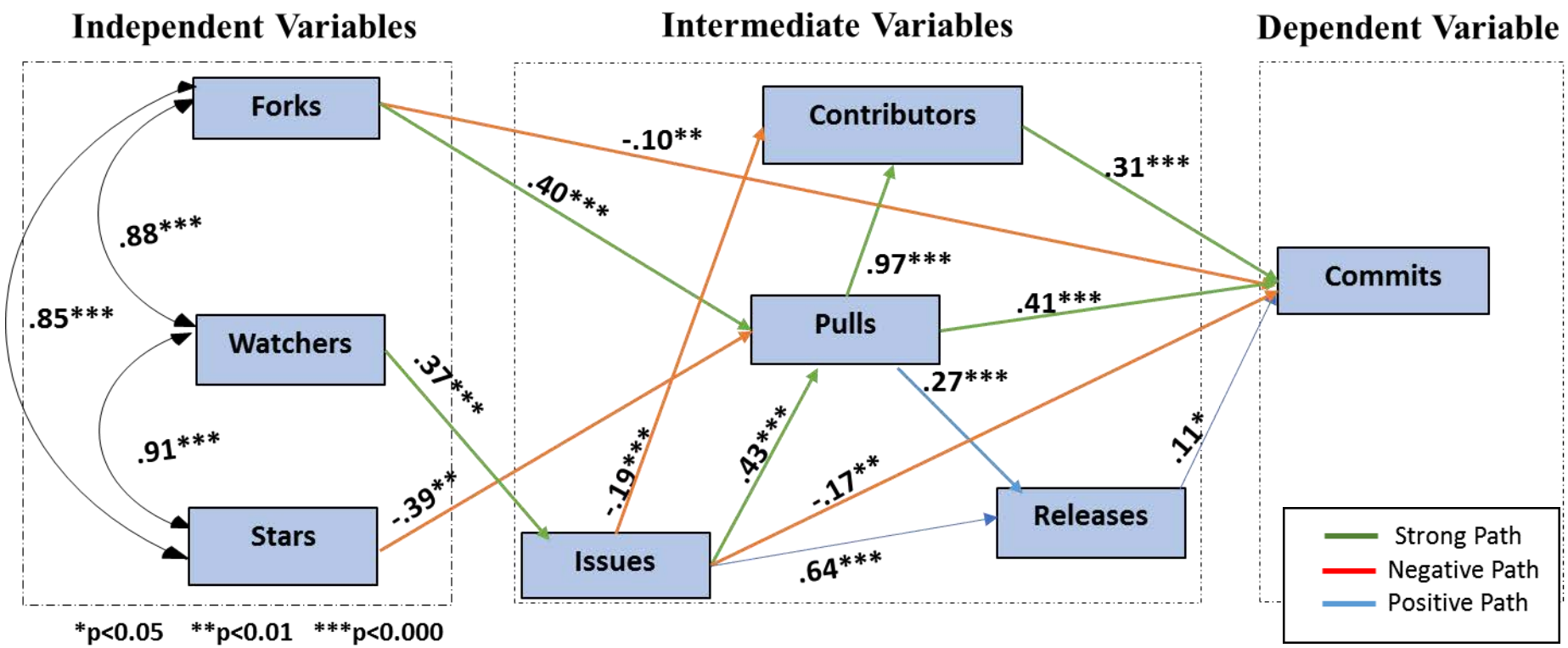
Relative Chi-Sq (X2/df)	Probability level (p)	RMSEA	TLI	CFI	GFI	AGFI
2.198	0.170	0.046	0.990	0.997	0.982	0.936

Figure 4-4: C++ Programming Language Path Model.



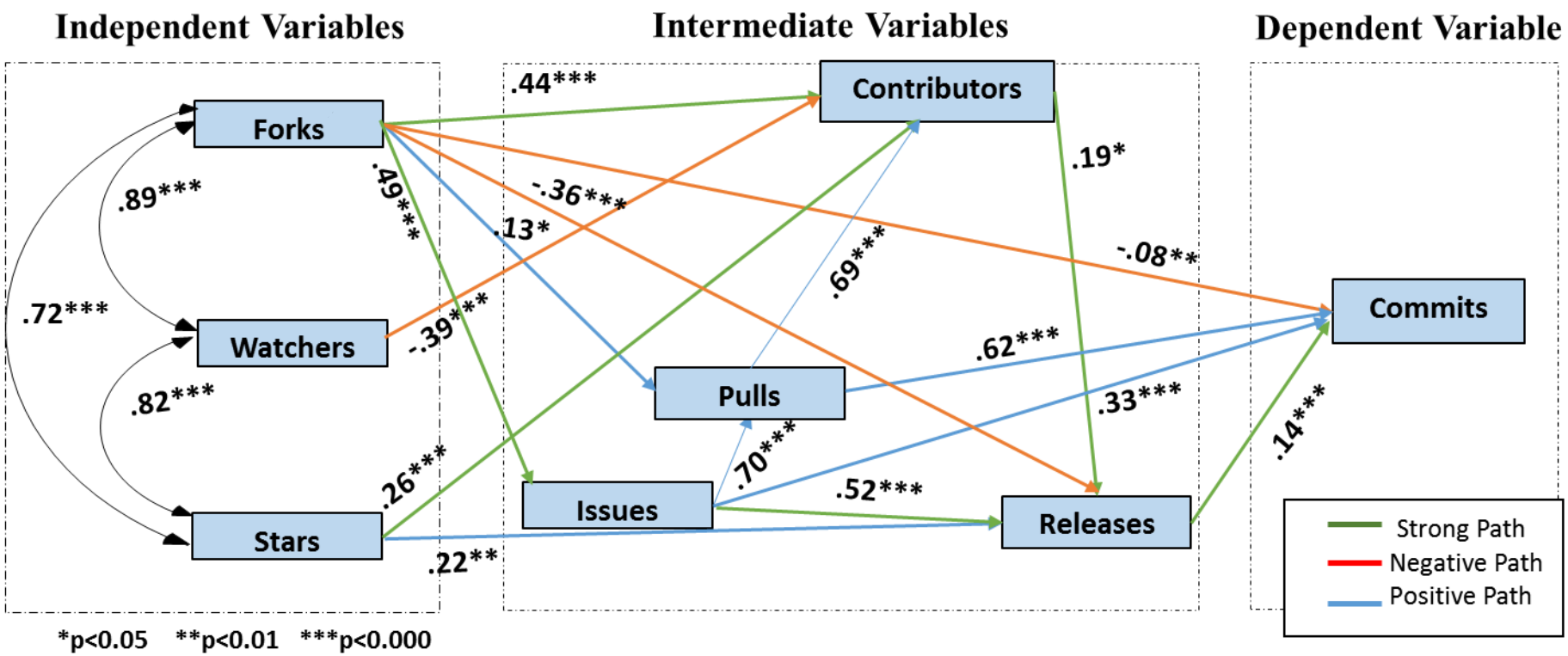
Relative Chi-Sq (X2/df)	Probability level (p)	RMSEA	TLI	CFI	GFI	AGFI
1.570	0.079	0.054	0.982	0.991	0.974	0.934

Figure 4-5: C# Programming Language Path Model.



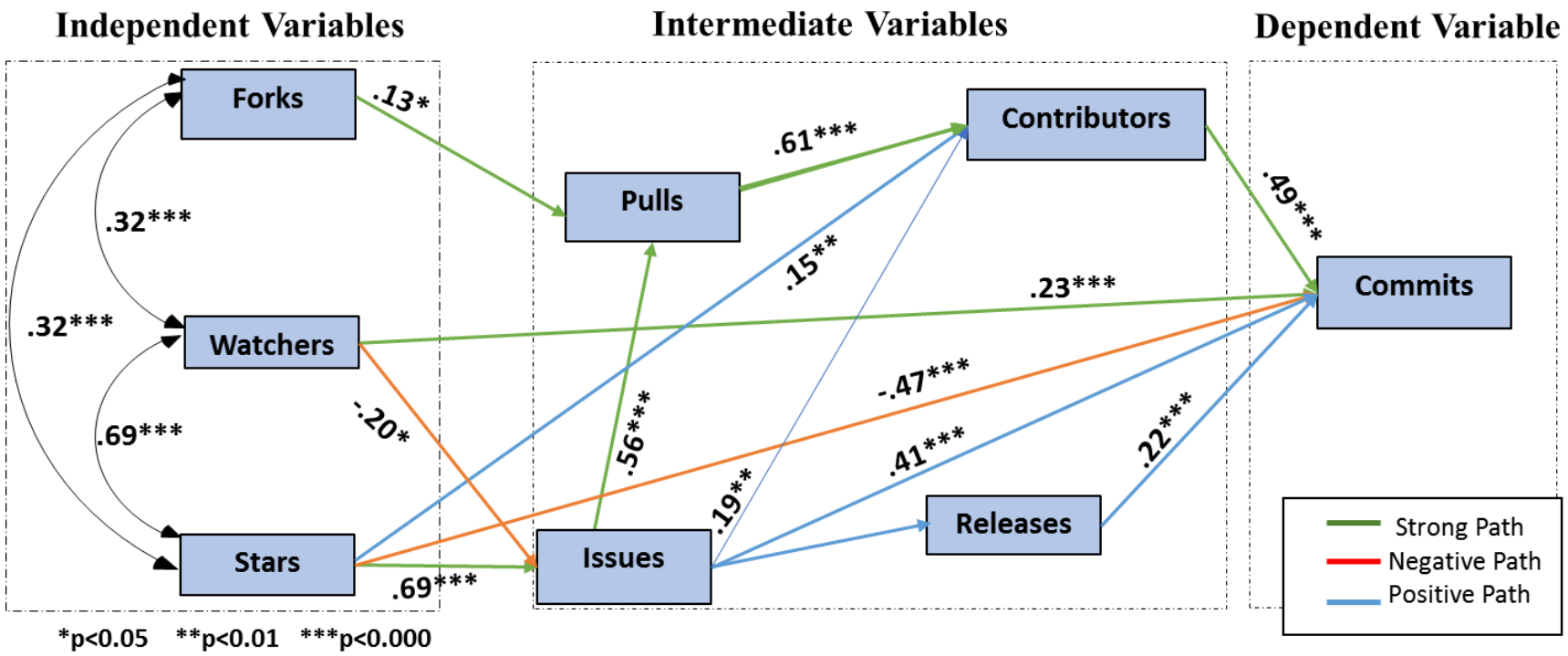
Relative Chi-Sq (X2/df)	Probability level (p)	RMSEA	TLI	CFI	GFI	AGFI
2.107	0.130	0.075	0.978	0.991	0.969	0.907

Figure 4-6: CSS Programming Language Path Model.



Relative Chi-Sq (X2/df)	Probability level (p)	RMSEA	TLI	CFI	GFI	AGFI
1.576	0.106	0.054	0.990	0.996	0.980	0.929

Figure 4-7: PHP Programming Language Path Model.



Relative Chi-Sq (X2/df)	Probability level (p)	RMSEA	TLI	CFI	GFI	AGFI
1.357	0.178	0.047	0.984	0.993	0.976	0.929

Figure 4-8: Ruby Language Path Model.

All GitHub language models display structural model differences in path strengths and overall pathways solutions. Values for the Standardized Total Effects of each of the key GitHub measurement constructs for all the above programming languages illustrated in appendix A. Analysis of the two phases of this study (pilot study and path model) is provided in Chapter Five.

4.3.2 Models Validation

For each programming language, small dataset (200 repos) were collected, this dataset cannot split into confirmation (usually 60% of dataset) and validation (40% of dataset) groups, so each model have been validated using a bootstrapping of 200 times and check validity using the Bollen-Stine p value (Cunningham, 2008; Hair, 1998). Table 4-12 shows validation results of the eight models.

Table 4-12: Bootstrap validation values for eight programming language models

Model	Bollen-Stine P
JavaScript	.527
Python	.547
Java	.333
C#	.428
C++	.607
CSS	.139
PHP	.652
Ruby	.194

CHAPTER 5

DISCUSSIONS

This Chapter discusses the research insights and its implications concerning both phases of the thesis repo work described in Chapter 4.

5.1 Phase one- Pilot study

The Pilot study was concerned with the top 10 (most Forked) GitHub repos for the top three OSS GitHub programming languages and observe the development of these repos for a period of 90 days with bi-weekly trend analysis blocks. The results of the Pilot study indicate that there likely is an ongoing culture of OSS development within those active GitHub repos.

This suggests other active GitHub repos might also exhibit substantial OSS development activities. Moreover, the analysis indicates different development methodologies and OSS programming languages are likely chosen by developers to tackle different OSS repo types. Hence, to pursue an overarching way to generate greater GitHub repo activity levels, this study selected the most utilized GitHub OSS programming languages, and then structural path models are constructed from the top 200 repos of each OSS language against GitHub elements.

5.2 Phase two - Structural path analysis study

5.2.1 SEM structural paths

The SEM structural path model of each of eight GitHub OSS languages, initially consisting of 200 repos per language, and so the total number of repos sampled was

1600. One unique high-quality model is delivered for each language with all relevant fit indices being excellent and further supported by a 200 times bootstrap validation (Dabbish et al., 2012; Hair et al., 1998).

A strong path represents one that significantly occurs in 5 to 8 of the GitHub OSS languages, it however does not represent the beta-weight strength of that path. A positive path occurs in 1 to 4 of the GitHub OSS languages. A negative path indicates a negative influence on the dependent path outcome construct.

The green arrows of Figures 4-1 to 4-8 represent the strongest significant causal pathways, and the red arrows show the significant negative pathways.

Looking at the JavaScript (JS) model, the green arrows illustrated in Figure 4-1 represent the strongest significant casual positive pathways. These positive pathways are: (1) Forks-to-Pulls-to-commits, (2) watchers-to-issues-to-commits, and (3) Stars-to-releases-to-commits. Hence, all three independent input constructs (Stars, watchers, and Forks) are important in generating the dependent construct (commits). Moreover, specific insight is as follows:

- Positive pathway (1) is logical – a Pulls request is a mechanism by which changes, and improvements are brought back into the original repo.
- Positive pathway (2) is logical – this indicates that watchers, regardless if they are contributors or not, are highlighting problems and requesting bug fixes because of using the repo after being notified about changes to it.
- Positive pathway (3) is interesting – it implies that increased popularity of JS repo ecosystems by the GitHub open source JS community has a direct impact on the number of releases for those repos.

The two red arrows represent medium significant negative pathways in the JavaScript model illustrated in Figure 4-1. The existence of these negative pathways in the JavaScript model indicates that: 1) more watchers alone do not favorably generate more contributors, and 2) increasing the Forks alone does not generate more releases. These pathways are backed up in the literature (Kalliamvakou et al., 2014; Padhye et al., 2014) indicating that not all watchers made a Pulls request and 40% of all Pulls requests do not merge.

Recently, JavaScript has seen considerable growth and language improvements(Jibaja et al., 2015). Common sense dictates that this must have a direct impact on the top JavaScript ecosystems considered in this thesis. Moreover, releases made by the top JavaScript repos are clearly a useful way to indicate active refactoring based on JavaScript growth and language improvements.

In Python programing language path model Figure 4-2, issues play a central role in the green pathways to commits. The strongest positive pathways are: (1) Forks-to-issues-to-Pulls-to-commits, and (2) forks-to-issues-to-releases-to-commits. The positive Python pathways are interpreted as follows:

- Positive pathways (1) increasing Forks mean more user interest in taking a copy of the repo, the probability of issues discovering high as more people read and use the repos, and this is expected since Python is a teaching language(Fangohr, 2004) as well as it is a first prototyping language, accordingly such repos will generate issues more than other non-teaching repos. Logically users will send a Pulls request which directly increasing the

commits.

- Positive pathways (2) In same way increase Forks result indirectly to issuing new releases which implies an increase in commits.
- Positive pathways (3) Increase reop popularity (starts) will have same effect as increasing forks, more stars mean more issues will appears and more people will send a pull request the repos owner.

The two red arrows represent medium significant negative pathways in Python structural model Figure 4-2, the existence of these pathways indicate (1) Increasing watchers negatively impact on issues, as stated before, watching repo never mean that user keen to fix issues or reporting one, they only want to keep watching what happened in repos. (2) Contributors are not the driving force for releases, as seen from the negative path from contributors towards releases, this is logical as the number of contributors does not directly affect releases.

The strongest significant causal positive pathways for Java programing language model Figure 4-3 represented in green arrows are: (1) watchers-to-commits, (2) Stars-to-issues-to-commits, (3) Stars-to-issues-to-Pulls-commits. Stars and watchers as independent constructs are important in generating the dependent construct (commits). Java programming languages used in mobile software applications, resulted in the positive pathways for Java interpreted as follows:

- Positive pathway (1) the watchers directly influence commits, when the user watchers any repos he gets a notification about what happened inside the repo. Researchers (Badashian & Stroulia, 2016; Borges et al., 2016b) used watchers as a measure of how good repo, they indicated that it has a

similar influence as Stars. Receiving repo notification encourage the user to commits (i.e. a bug fix).

- Positive pathway (2) implies that increased repo popularity impacting issues, more people interest in repos logically lead to more bugs and issues discovers, which in return leads to generating more commits.
- Positive Pathway (3) is logical too as repo gets more popular, issues increased, thus Pulls (open and close) will increase too, that generate more commits.

Medium significant negative pathways in Java structural model Figure 4-3 , represented in red arrows, The negative path in Java are: (1) forks-to-releases and (2)stars-to-commits, These two negative paths indicates that: (1) Forks does not generate new releases, usually, more Forks mean more people interest in repos , and potentially they contribute or increasing issue but in Java programming Language which is used to make the library and other functional applications , users reuse the code without contributing or issuing any bugs (Badashian & Stroulia, 2016). (2) Increasing the Stars alone does not generate more commits. Stars and commits are weakly related according to (Borges et al., 2016b) so it makes sense that increasing Stars have small influences in commits.

Green arrows in C# path model Figure 4-4 represents the strongest positive pathways, these positive pathways are: (1) Forks-to-issues-to-Pulls-to-commits, (2) Forks-to-contributors-to-commits and (3) Stars-to-contributors-to-commits. Hence, two

independent constructs (Forks and Stars) are important in generating the dependent construct (commits). Possible insight into the positive pathways are:

- Positive pathway (1) a Fork is a copy "clone" of the repo, the probability of issues discovering will be high as more people read and use the repos, logically users send a Pulls request which directly increasing the commits.
- Positive pathway (2) In same way increase Forks result indirectly in issuing new releases which will lead to an increase in commits.

The red arrow represents medium significant negative pathways in the C# model. The existence of the negative path in the C# model indicates that: (1) more watchers alone do not favorably generate more contributors because there are many users of the repos that are clearly not contributors.

C++ green positive pathways (Figure 4-5) are: (1) Forks-to-Pulls-to-commits, (2) Stars-to-Issues-to-Pulls-to-commits, and (3) Stars-to-issues-to-commits. The two independent constructs Forks and Stars are important in generating the dependent construct (commits). C++ is used for low level computer graphics, libraries, framework, kernel drivers and operating system(Schmidt et al., 2013). Analyzing C++ positive pathways shows that:

- Positive pathway (1) increasing Forks effect Pulls, increased opportunities for community engagement, and decreased time to incorporate contributions.
- Positive pathway (2) Stars play role in increasing the issue which logically causes an increase in commits.

The two red arrows represent medium significant negative pathways in the C++ structural model illustrated in Figure 4-5. The existence of these negative pathways in the C++ model indicates that: (1) Watchers only keep the user updated about what happens in repos, that does not mean the watcher potentially will be future contributors in C++ languages. (2) increasing the Forks alone does not generate more commits.

For CSS programming language green positive pathways Figure 4-6 are: (1) Forks-to-Pulls-to-commits, (2) Forks-to-Pulls-to-contributors-to-commits, (3) watchers-to-issues-to-Pulls-to-commits. Forks and Watches are two independent constructs most important in generating the dependent construct (commits). CSS is a web domain languages (Ribeiro & da Silva, 2012). CSS path models have similar strong paths as Java scripts were:

- Positive pathway (1) increasing Forks will increase Pulls, which in turn increase opportunities for community engagement, commits will be increased in turn which makes sense as accepted Pulls request will incorporate more contributions, in web languages, developers tend to fork software in order to participate and fix bugs which in turn generated more Pulls request that merged.
- Positive pathway (2) is a logical pathway where increasing Forks will lead to increase in Pulls request which in turn maximize the opportunity for community engagement (Kalliamvakou et al., 2016) thus more commit will occur.
- Positive pathway (3) implies that increased watchers will indirectly lead to increasing commits via increase issues and Pulls, the role of watchers here is

not only to reuse the software, but they actively participate in bug fixing, or bugs discovering, so mainly they are active users who want to participate in repos not just watching what going on across the repo.

The four red arrows represent medium significant negative pathways in the CSS structural model illustrated in Figure 4-6. The existence of these negative pathways in the CSS model indicates that: (1) Forks alone do not favorably generate more commits, (2) Stars alone does not generate more Pulls. And (3) Issues negatively impact on contributors and commits.

Green positive pathways in PHP programing language Figure 4-7 are: (1) Forks-to-contributors-to-releases-to-commits, (2) Stars-to-contributors-to-releases-to-commits, and (3) Stars-to-releases-to-commits. Thus, Forks and Stars are important in generating the dependent construct (commits). Moreover, possible insights are:

- Positive pathway (1): increase in Forks will result in increased community engagement, which in turn increase releases thus commits will increase too, this is a logical pathway.
- Positive pathway (2): More popular repos naturally lead to more contributors and will be these will encourage more releases which in turn increase commits, generally speaking, new releases, will cause acceleration in Stars repos get (Borges et al. 2016a)
- Positive pathway (3): Stars will absolutely affect releases, each new release will encourage more commits.

The two red arrows (medium significant negative pathways) in Figure 4-7 for PHP structural paths models indicate that: (1) more watchers alone do not favourably generate more contributors, and (2) increasing the Forks alone does not generate more commits or more releases. Also, PHP shares similar positive and negative paths as JavaScript and shares similar negative paths as CSS language as both consider a web-page design programming language (Nixon, 2014).

Ruby green positive pathways (Figure 4-8) are: (1) Forks-to-Pulls-to-contributors-to-commits, (2) Stars-to-issues-to-Pulls-to-contributors-to-commits and (3) Stars-to-issues-to-releases-to-commits. All three independent constructs are important in generating the dependent construct (commits). Possible insights are:

- Positive pathway (1): Forks affect community engagement and results in more contributors to repos (Vasilescu et al., 2015; Jiang et al., 2017), Thus resulted in increasing commits.
- Positive pathway (2): More popular repos are more contributors will be these will encourage more releases which in turn increase commits, generally speaking new releases will cause acceleration in Stars repos get (Borges et al., 2016b)
- Positive pathway (3): Stars affect releases, each new release will encourage more commits.

The two red arrows represent medium negative pathways for Ruby structural path model (Figure 4-8). These negative pathways indicate that: (1) Watchers alone do not positively generate more issues, and (2) increasing the Stars alone does not generate more commits. Ruby tends to have a similar path to C++, C# and Python.

5.2.2 SEM structural path comparison

The eight SEM path models are compared. The ***frequency*** of each significant path is displayed in Table 5-1. For example, issue-to-Pulls path considers strong path because its frequency is eight. That means this path exists in all eight language models. Whereas, the path from watchers-to-commits has a frequency of two, thus it is considered a weak path because it exists in two language path models which are Java and Ruby.

Hence. Table 5-1 adopts three classification levels depend on path frequency: weak (1-3), moderate (4-5) and strong (6-8) to better understand the causal pathways across all eight structural path models. This allows all eight SEM structural path models to be grouped by relative importance. The "*path type" column* in Table 5-1 indicates if the path are positive and/or negative for each programing language models. The "*path model" column* shows for which programing language models does the path exist. The red colour indicates when a negative path exists.

From Table 5-1, the issues-to-Pulls path appears in all eight programing language models, thus it considers a strong path. Issue used as tracker in GitHub, Issues used to report bugs, enhancements and tasks (Gousios, 2013) More issues in repos implies more collaboration from developers, GitHub is a collaborative development (Lima et al., 2014) thus increasing issue will results in more developer offer bug fixing, adding new features or , this done via sending Pulls request (Kalliamvakou et al., 2014; Padhye et al., 2014; Tsay et al., 2014; Kalliamvakou et al., 2016).

Another strong path could be seen in Table 5-1 is Pulls-to-contributors, it appears in all model, this path is logical, as a Pulls request considered a way for getting more people to engage in community and contribute to code (Kalliamvakou et al., 2016).

Also, releases-to-commits is a strong path occurs in all models, not too many searchers search in that path, the existence or a relation between releases and commits is logical, adding a new release to repo results in more commits about that releases.

In GitHub bug reporting, user feedback or even suggestion of new feature in software or improving documentation these all considered as feedback for repos owners, these feedback expressed as new issues (Liao et al., 2018), when issue increased repos contributors will issuing a new releases the new releases overcome or solve the issues mentioned in user feedback, so path from Issues-to-releases, is logical and strongly occur in seven programming path models.

Pulls-to-commits path occur in seven programming languages(this path not found in Ruby language only), according to (Kalliamvakou et al., 2014; Tsay et al., 2014a) Pulls and commits should be balanced, increasing Pulls will potentially increase contributors.

The moderate and strong paths in Table 5-1 are then combined into one general GitHub ecosystem path model Figure 5-1 (in Section 5.4). This visualizes the commonalities across programming languages and allows OSS developers to gauge

how (what place), where (what construct elements), and when (what stage) they generate further repo activities.

Table 5-1: The appearance of GitHub elements SEMs path for eight programming languages.

GitHub Sub- Path	Frequency	Path Strength	Path Type	Path Models
Issues-Pulls	8	Strong	Positive	JavaScript, Python, Java, C#, C++, PHP, Ruby, CSS
Pulls-Contributors	8	Strong	Positive	JavaScript, Python, Java, C#, C++, PHP, Ruby, CSS
Releases-Commits	8	Strong	Positive	JavaScript, Python, Java, C#, C++, PHP, Ruby, CSS
Issues-Releases	7	Strong	Positive	JavaScript, Python, Java, C++, PHP, Ruby, CSS
Pulls-Commits	7	Strong	Positive	JavaScript, Python, Java, C#, C++, PHP, CSS
Forks-Pulls	5	moderate	Positive	JavaScript, C++, PHP, Ruby, CSS
Stars-issues	5	moderate	Positive	Python, Java, C#, C++, Ruby
Stars-Contributors	5	moderate	Positive	JavaScript, C#, C++, PHP, Ruby
Issues-Commits	5	moderate	Positive	JavaScript, Java, C++, PHP, Ruby
Contributors-Commits	5	moderate	Positive	Java, C#, C++, Ruby, CSS
Watchers-Issues	4	moderate	Positive/Negative	JavaScript, Python, Ruby, CS
Issues-Contributors	4	moderate	Positive/Negative	Python, Java, Ruby, CSS
Contributors-Releases	4	moderate	Positive/Negative	JavaScript, Python, Java, PH
Forks-Contributors	4	moderate	Positive	Python, C#, C++, PHP
Forks-Issues	3	Weak	Positive	Python, C#, PHP
Pulls-Releases	3	Weak	Positive/Negative	C#, C++, CSS
Watchers-Commit	2	Weak	Positive	Java, Ruby
Stars-Releases	2	Weak	Positive	JavaScript, PHP
Stars-Commits	2	Weak	Negative	Java, Ruby
Stars-Pulls	2	Weak	Negative	C++, CSS
Forks-commits	3	Weak	Negative	C++, PHP, CSS
Forks-Releases	3	Weak	Negative	JavaScript, Java, PHP
Watchers-Contributors	4	moderate	Negative	JavaScript, C#, C++, PHP

Three of the programming Languages (JavaScript, Python and PHP) share the same application domain, web-based application. These three programming languages have

similar pathways with minor's differences (Borges et al., 2016b). Those paths are as follows:

- Pathways from forks towards commits via Pulls and contributors, appears in all three language models. The users of a forked repo make suggestions back to the original repo via Pulls requests. A Pulls request use for bug fixing, adding new features, and/or other types of modification. Pulls requests increase opportunities for more contributors to be engaging in repo community (Kalliamvakou et al., 2016). Accepting a Pulls request results in adding a new contributor. Rejecting it causes a Pulls request to be closed (Kalliamvakou et al., 2014; Padhye et al., 2014; Tsay et al., 2014a). Web page application requires cooperation from inside and outside developer team. End user usually play role in the development of these application. User feedback and recommendation for better performance seriously considered and encouraged by the development team. That is a logical part of the development process for such applications.

  As a web application (e.g. Facebook, and other social media apps) the popularity of such repos plays a central role in advancing those software ecosystems towards the generation of more issues and releases. Popularity is measurable by the number of Stars and Forks (Borges et al., 2016b).

- The path from watchers-to-issues appears in JavaScript and CSS, indicating that watchers are highlighting problems and requesting bug fixes be added when engaging such repos after being notified about most recent changes. At same time, watchers do not appear to be involved in any contribution, so they remain as passive users of the repo.

- The negative path from Forks to releases to commits appears in JavaScript and PHP, while there is no path between Forks and releases in the CSS language. Releases do not increase according to Fork. PHP and CSS programming languages both have a negative path from Forks to commits. Forks indicate the user is interested in the repo, and Forks is a measure of repo reuse (Badashian & Stroulia, 2016).

The other five programming languages (Java, Python, C#, C++ and Ruby) used for various other application including mobile software applications, different library software. They tend to display similar path models. In such an application domains user usually forks their own local copy to reuse software (Badashian & Stroulia, 2016). User Forks and modify their own local copy. Usually, user will be discovered issues and report them to repo developing team. Common Pathways are:

- Four programming Languages (Python, C#, C++ and Ruby) have directed or indirect path from forks via pulls to commits.
- Pathways from stars-to-issues-to-pulls-to-contributors-or directly to-commits, this is logical as stars indicates user interest in such repos (Vasilescu et al., 2015). Stars-to issues path appears in all five models. Popular repos get more attentions and user will actively participating in finding bugs and fix it, more user interest mean more issues will discovered and more Pulls which will lead to increase in contributors.
- Path (3) appears in JavaScript and PHP this is interesting as it implies that increased popularity of the repo ecosystems by the GitHub open source community also increases community engagements, which in turn, help in issuing more releases for the repo, thus increase commits.
- The negative path from watches to issues, or to contributors, is expected as the application domain in this case can affect. Users want to reuse proper copy of the repos so they keep watching what will happen to software and how it is being developed (passive watchers) so they could forks their own copy when they think the code is ready.

5.3 Standardized total effects model's comparison

The standardized total effects of each language structural model are presented in Appendix A as Tables A-1 to A-8. These allow the comparison of each construct onto its commits level of activity into the specific OSS programming language's repos.

Table 5-2: Standard total effects for Commits (across all eight OSS programming languages).

Programming Language	Data Set	Watchers	Stars	Forks	Issues	Pulls	Contributors	Releases
JavaScript	**195**	0.32	0.06	0.03	0.74	0.5	0.04	0.16
Python	**195**	-0.19	0.15	0.23	0.54	0.79	-0.03	0.15
Java	**170**	0.31	-0.10	-0.02	0.72	0.41	0.30	0.15
C++	**195**	-0.07	0.16	0.19	0.55	0.70	0.20	0.21
C#	**199**	-0.07	0.20	0.17	0.49	0.69	0.19	0.30
CSS	**199**	0.21	-0.27	0.17	0.58	0.70	0.31	0.11
PHP	**197**	-0.01	0.04	0.38	0.84	0.63	0.03	0.14
Ruby	**165**	0.07	0.15	0.04	0.78	0.30	0.49	0.22

The standardized total effects of each construct on its commits level of activity into the OSS programming repo are shown in table 5-2. All eight language data sets exceed 150-160 cases (Hair et al., 1998; Muthén & Muthén, 2002).

This suggests OSS repo creators should continually promote new issues and new pulls as a motivation to draw in OSS developers and users and to request that they also communicate any significant issues they find to the repos community of contributors.

The major GitHub OSS constructs contributing to generating commits are Pulls and issues as highlighted in Table 5-2. This is a motivation for OSS developers and users communicating arises when significant issues appear.

In Table 5-2, a five classification levels are defined: negative impact is represented by orange cells, very small effect is shown as (<0.1) grey cells, small effect (0.1 - 0.3) are a blue coloured cells, moderate (0.3-0.5) are shown as yellow cells, and green cells represent the largest effects (>0.5).

Open issues encourage repo communications. These can lead to increased action towards solving the issue by encouraging developers to investigate the issue. Hence there is a logical connection between Pulls, issues and commits. The literature supports that established issues can generate considered pulls requests, and this is a measure for increasing user engagement and for drawing in more OSS developers (Kalliamvakou et al., 2014; Padhye et al., 2014; Tsay et al., 2014).

The Ruby programming language also shows issues exerting a big influence on commits. In the Python, C#, C++ and PHP models, watchers negatively impact commits overall. This indicates that developers in those ecosystems are watching the repo for changes and subsequently hold back progress in some way. Watchers are more involved in repo issues than code changes (Sheoran et al., 2014).

In all models, the number of Stars provided to the repo generates a very small contribution towards commits, as more Stars alone do not generate further additional OSS repo commits.

Stars and Forks can sometimes indicate a reluctance to contribute positively (see Table 5-2). Forks can generate new ideas, create a new code, and then draw some of the original repo developers into a new software sub-ecosystem. Thus, retarding the original repo's activity level. Thus, Forks are needed, but they can also be detrimental.

Multiple intermittent and minor version releases exert less GitHub repo commits levels because they often involve slight improvements, and only require minimal activity level contributions and then the OSS developers interest in the repo codes. Also, numerous revision releases do not necessarily draw in contributions from additional quality OSS developers.

5.4 GitHub programming languages summary

GitHub's JavaScript, Python, Java, C++, C#, CSS, PHP and Ruby path models provide an understanding of the different significant developer contribution pathways towards raising the repo's commits. The significant pathways offer the repo's creator decision-making capabilities that can be used to trigger faster repo software development through to its next completion point.

The repo's commits level is a performance benchmark that is now available to the GitHub repo creator. This approach can benchmark against the competition. This

approach is also behavioural, and where the developer ecosystem's responses lift the repo's OSS developer consumption and satisfaction. Its values deliverance processes are also enhanced. Thus, a consumptive-values approach provides a future pathway towards tapping big data sources and to also delivering real business(Hamilton & Tee, 2015).

Commits are key direct drivers of the repo's productivity and activity. Other contributors are further indirect drivers. Since Pulls and issues are the strongest drivers of the repo's productivity and activity, this suggests creating high levels of issues and Pulls requests should be a prime target consideration for repo creator's core team of developers.

Reviewing all path analysis models for the eight programming languages in Chapter 4 indicates the existence of a connectivity pattern among model elements. Figure 5-1 represents the proposed GitHub ecosystem generalized path model showing positive and negative connectivity. The connectivity's are now classified as follows: (6-8) represent a strong connectivity illustrated as the green paths, these connections appear most (6-to-8) of the path models, (4-to-5) represents moderate as blue paths, and (-4) red negative path.

Considering only strong and medium connectivity's the general ecosystem path model for GitHub programming languages is proposed as Figure 5-1. This general ecosystem path model likely applies to other GitHub programming languages not included in this thesis.

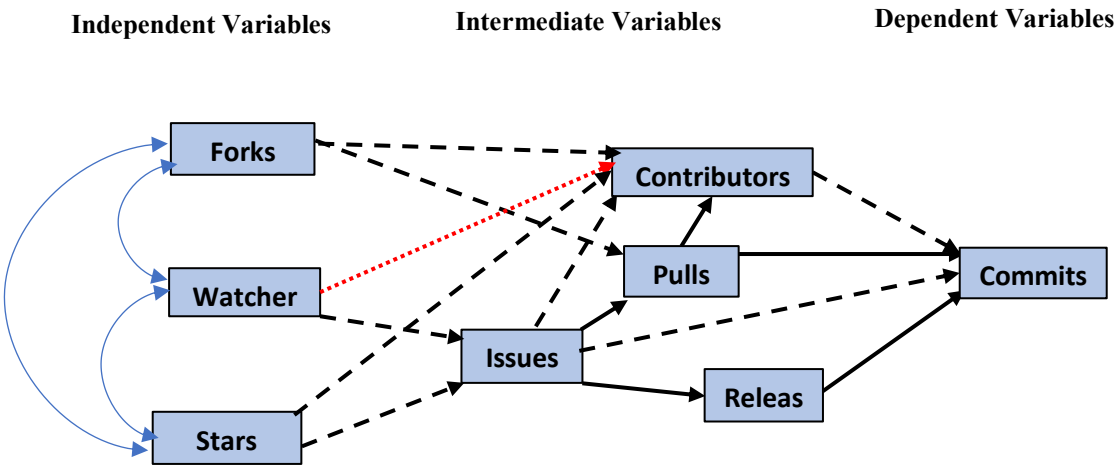


Figure 5-1: A generic Path Model for GitHub.

5.5 Summary

GitHub is an OSS development site, housing many repos. Each repo normally chooses to adopt one of many languages. The eight most important (by Forks) GitHub languages are JavaScript, Python, Java, C#, C++, CSS, PHP, and Ruby. This study's eight model SEM path model investigation deduces the generalized structure of a top GitHub ecosystem. It also deduces that a likely way to grow a repo's activity level is to raise all possible issues that emerge and do so in a sequential manner that likely encourages OSS developers to continuously generate and complete additional Pulls requests.

There are three strongest pathways leading to commits. One pathway links Stars and issues to pulls then commits, for other pathway combines Forks to Pulls, then to contributors, then commits. The last of these strong pathways is from Stars to issues then to releases to commits. These green pathways (Figure 5-1) represent the strongest paths of influence capable to increase the GitHub repo commits.

This process speeds-up overall repo development and possibly lowers the overall completion time and cost of development. Stars, watchers, and Forks are independent variables used as the input constructs for all SEM models as these constructs starts the repos OSS development. Issues, pulls, releases and contributors are intermediate constructs that help build the OSS repo solution. Commits is the measure of the driving for OSS repo solution. Large numbers of commits represents likely repo success and sustainability (Xavier et al., 2015). This study explored the possible paths and pathways that can affect commits which in turn can affect the ecosystem.

From the eight SEM models, Pulls and issues are the game players affecting GitHub repo ecosystem, whilst releases and contributors have small effects. Watchers have a negative impact on repo activity.

CHAPTER 6

CONCLUSIONS

This Chapter provides a discussion of different GitHub case studies provided in Chapter 4 and 5. The pilot study suggested trends may exist in GitHub repos, Chapter four path modelled eight programming languages confirming the existence of a GitHub ecosystem.

6.1 Current Implication of Research

This GitHub study follows responder behavioural patterns, Information Integration Theory, and the Theory of Social Translucence. This framework allows behavioural activities to be gauged collectively and measured against each repo's overall activity level. This allows a new way to compare repos and to understand repos once the masking features such as: size, programming-language, degree-of-complexity and longevity are removed.

6.1.1 Theoretical Implications

This GitHub study follows responder behavioral patterns, in particular - Information Integration Theory, and the Theory of Social Translucence. This framework allows behavioral activities to be gauged collectively and measured against each repo's overall activity level. This allows a new way to compare repos and to understand repos once the masking features such as: size, programming-language, degree-of-complexity and longevity are removed.

Extensions to this study can map each repo responder's/collaborator's identity, contributions, and ongoing activities through to GitHub repo followers, watchers and stars-provided into their social interaction domains including Facebook, websites, Twitter, and Wikis (Aggarwal et al., 2014). Here, interpretations of value by understanding social network site consumer engagements (Hamilton & Tee, 2013) can be incorporated to extend the behavioral understanding of GitHub's social and external responders.

This study reviews the ecosystem of software development which can then supplement processes involved in software engineering and development. It also extends to the concept of real-time social interactions - such as the understanding behaviors of humans and their representative avatars in real world gaming.

6.1.2 Practical Implications

Accessing GitHub repos to extract data is a time-consuming process, for each repo I count the number of committers, commit and extract the 10 GitHub elements used in this thesis. Retrieving data from GitHub is limited to 30 access/hour for non-GitHub member and 6000 access per hour to members with access right, this process impacts the time required for dataset collection.

The activity level of JavaScript, Python, Java, C++, C#, CSS, PHP and Ruby repos responders is measured using repo-collated measures. These behavioural measures first include pull-requests and Issues which results in subsequent commit changes. Pull-requests impact on repo contributions and on repo version releases, and positively

influence on commits. Commit changes are generally clarified through comments with linkages into repo documentation.

A lead focus for the repo creator and the core team of collaborators is to generate additional commits. Here, commits can be encouraged by cross-promotional strategies including: (1) encouraging pull-requesters to respond and to generate multiple commits, (2) promoting the starring of the ongoing value of the repo's development on Facebook, Twitter, and web media, and also converting social media watchers into pull requesters, and (3) engaging developer forums, Wikis, conferences and across other social connectivity avenues directly targeted towards encouraging more pull requests and follow-up commits.

Social media sites can also add transaction-related repo information via inclusions of community 'fan-pages.' Fan-pages help to build stronger communities, provided they show usefulness, economic value, and are suitably branded. Here promotions and/or other consumer benefits can be incentivized(Hamilton & Tee, 2013).

In addition, to further highlight and draw developer traffic, fan-pages news can be linked to HackerNews and GitHub Explore(Borges et al., 2016). Ultimately the key internal approach is to generate very-rapidly reviewing and incorporating decisions across all commits.

A second behavioural approach is to recognize committers by crediting their contributions against their personal email. This is achievable by recognizing, ranking,

and promoting each contribution as enhancing: performance and/or quality and/or service and/or economic value and/or emotional perception (Hamilton & Tee, 2015). These value recognition triggers are rewards to the respondent committer, and they likely positively affect the committer's satisfaction and ongoing loyalty(Alshomali et al., 2017). This recognition approach behaviourally encourages the committer to pursue further opportunities of benefit to a GitHub project. It also enhances their personal profile, and it promotes more repo activity.

6.2 Future Implications and opportunities for Research

6.2.1 Measurement aspect

To further validate the repo ecosystem of GitHub JavaScript, Java, Python, C#, C++, PHP and Ruby structural path model additional studies are suggested (1) random sampling across the full suite of these languages, and (2) re-testing against each key GitHub programming language. (3) Large closely type-lined and similar software programs design area, top activities cases with a programming language.

The refinement of the pull request counts is another measurement consideration. Pull-requests occur because of internal commits for review as well as via forked releases of the original repo. Some forks-pulls-requests loop back into the originating repo. Hence, it may be useful to categories pulls-requests, and also to consider longitudinally if forks-pulls do actually occur later during repo development. This research is underway.

There remains a need to create and deploy APIs that monitor repo activity levels over time. This can expose where open source software development offers maximum improvement for the GitHub repo under consideration.

6.2.2 Theoretical aspect

GitHub OSSD studies can be theory-based, and/or behaviorally-based, and/or translucently-based, and/or values-based. They can also be linked via social networks and web media through into other consumer marketing and retailing approaches - typically focusing on consumer motivation, consumption and gratification aspects(Hamilton & Tee, 2015). In additional, the approach taken by this thesis could be operationalised as a process model to further understand software development processes, by either a design science research methodology or by an action research approach.

6.2.3 Management Aspect

The repo activity level model is applicable for GitHub JavaScript repo creators (and other seven programming languages studied in this thesis). It can be astutely managed to generate high repo level activities. It can be interpreted through Table A-1 total effects in Appendix A and Figure 4-1 in Chapter four, path strengths towards better targeting, and harnessing of a repo's reach, and engagement, across relevant software development communities.

Learning how to extract pertinent information from responder review comments is often useful to a repo originator seeking to improve ongoing repo deliverables.

Approaches to understanding big data vary, but Bello-Orgaz et al. (2016) describe big data social capture approaches are of use when considering GitHub's watchers.

Repos can be more closely managed by developing text capture routines to extract responder key words from GitHub documentation. For example, value(s)-related words epitomising behaviors can include motivation (intentions to act) towards engaging/actioning, consumptive actions being undertaken, and gratification reflections of actions delivered. This data can then be real-time analysed, thus keeping GitHub repo originators behaviorally attuned to individuals and to their core collaborators.

6.3 The Research Outcomes of this Thesis

The thesis observed GitHub repos to measure change factor in each repo, repos under study was chosen to depend on parameters that included: Eight programming languages, most forks repos, and repos with high Forks counters.

The path model approach is regression based it identifies the most important and least important constructs. However, it assumes data collection measures are made without random measurement error. This feature can disguise multicollinearity effects (Kalliamvakou, et al., 2016; Kline, 2015). In this thesis, I control for these multicollinearity effects by our research design.

The number of stars-provided to the repo make a lesser contribution. The forks work against the repo's progress by generating very minor negative total effects into the

repo's activity level. They sometimes dilute the focus of the repo's software development strategies. Here, a fork may generate new ideas, create a new repo, and then draw some original repo developers off into this new software development direction, thus retarding the original repo's activity level. Multiple intermittent and minor version releases exert less GitHub JavaScript repo activity levels because they often involve slight improvements, and only require minimal activity level contributions. More commits also bring more changes to documentation, and as a GitHub repo's activity level rises, additional documentation emerges as a continual repo requirement.

Commits are key direct drivers of the repo's activity level; other contributors are indirect drivers of the repo's activity level. Pulls and commits are the strongest drivers of the repo's activity level. This suggests creating high levels of pull requests should be a prime target consideration for repo creator's core team of developers. This study offers a big data direction for future work. It allows for the deployment of more sophisticated statistical comparison techniques. It offers further indications around the internal and broad relationships that likely exist between GitHub's big data and models linking through to business/consumer consumption, and how these may be connected using improved repo search algorithms to releases business value. Hence, the research questions of this thesis are answered as follows:

RQ1: What elements are present in the GitHub OSS ecosystem?

Answer: The GitHub ecosystem consists of at least eight key elements (star, fork, watch, issues, contributors, releases, pulls and commits) as shown in the Figure 3-4 conceptual model.

RQ2: Do programming languages show different path models in the GitHub ecosystem?

Answer: Different programming language platforms the GitHub OSS ecosystem display different paths in their respective ecosystem - as evidenced in the SEM path models of Figures 4-1 to 4-8.

RQ3: What relationships exist between each element when affecting the commits in the GitHub ecosystem?

Answer: Tables 5-1 and 5-2 summarize the complete relationships between the elements of each GitHub OSS programming language in the the GitHub OSS ecosystem. It is noted there are multiple relationship pathways that contribute towards the commits. These complex relationships show differences in their contributions towards commits amongst the top eight GitHub OSS programming languages examined in this research.

RQ4: How does each element influence the GitHub ecosystem?

Answer: Table 5-1 and Figure 5-1 together explain the generic path model for the GitHub OSS ecosystem. Table 5-1 shows the relationship can be strong, moderate or weak, as well as either positive or negative. Figure 5-1 shows which elements generally produce a strong, positive, or negative path influence towards commits. This allows GitHub developers to focus on the elements that are key drivers capable of inducing and accelerating OSS development activities. For example, key initial elements affecting the general ecosystem for GitHub are: forks, issues, and pulls, whilst releases and contributors have smaller secondary effects, and watchers

generally have a negative impact because they are generally passive and typically remain outside the GitHub ecosystem community.

6.4 Practical conclusion

From the work done in this thesis, the following practical conclusion is drawn: As far as my reading to research in GitHub key elements this is the first study that takes in consideration eight GitHub elements (actually it is ten if calculating issues as (issue open and close, and Pulls same as pulls open and close), The main finding was that forks is the most important key elements that could help in making repos more popular and more active. This thesis statistically proved the weight each element affects the commits, were fork is the most influencing one followed issues and pulls. The more external developers fork a repo, the more commits which in turn increase the opportunities to:

- Increase number of repo developers;
- Fixing more bugs and error in repos; and
- Progressively update documentation.

Accordingly, a recommendation for a successful repo, is to consider forks count carefully when build your repos to seduce more user to forks it, by carefully selecting a programming language, make documentation clear and your code should be easy to understand.

Each programing language has different path model, but the path with a fork, pulls or issues commonly found in most of them. In this thesis, I used to test and evaluation datasets both have been written in very well-known and popular programming

languages as well as have most forks counters. Most forks repos collected in datasets also have most stars which in turn prove that forks effects users and encourage them as a result to participate in repos and start it.

Not all repos collected for case studies in this thesis were valid, applying a condition on repos helped in eliminating outliers (Goyal et al., 2018) . Repos with unbalance commits forks ration should be eliminated as this repos may affect the final results, such repos are questionable and when tracing the eliminated repos back, it was obvious that it is outliers, for example shadowsocks / shadowsocks repo in Python language has very high rank as most forked repos as well as has very high stars count and considered one of most popular Python reops for theses stars and forks count but the unbalance contributors commits make it in questionable, these repos appear to be banned and it is illegal (hacker) application.

Extensions to this study can map each repo responder's / collaborator's identity, contributions, and ongoing activities through to GitHub repo followers, watchers and stars-provided into their social interaction domains including Facebook, websites, Twitter, and Wikis(Aggarwal et al., 2014). Here, interpretations of value by understanding social network site consumer engagements(Hamilton & Tee, 2013) can be incorporated to extend the behavioural understanding of GitHub's social and external responders.

Main challenges when extracting data from GitHub is time-consuming. To reduce data collection time, I recommend using AUTH offered by GitHub which extend the amount of retrieved data each time.

REFERENCES

Aggarwal, K., Hindle, A., & Stroulia, E. (2014). *Co-evolution of project documentation and popularity within GitHub.* Paper presented at the Proceedings of the 11th Working Conference on Mining Software Repositories. (PP. 360-363). ACM. Hyderabad, India.

Alshomali, M. A., Hamilton, J. R., Holdsworth, J., & Tee, S. (2017). GitHub: Factors Influencing Project Activity Levels. In: Proceedings of the 17th International Conference on Electronic Business, pp. 116-124. From: ICEB 2017: 17th International Conference on Electronic Business, 4-8 December 2017, Dubai, United Arab Emirates.

Amir, M., Khan, K., Khan, A., & Khan, M. (2013). An Appraisal of Agile Software Development Process. International Journal of Advanced Science & Technology, 58(56), 20. Sydney, Australia. Doi:10.1.1.398.1625.

Andersen-Gott, M., Ghinea, G., & Bygstad, B. (2012). Why do commercial companies contribute to open source software? International Journal of Information Management, 32(2), 106-117.London, United Kingdom. Doi:10.1016/j.ijinfomgt.2011.10.003

Anderson, M. J. (2001). A new method for non-parametric multivariate analysis of variance. Austral ecology, 26(1), 32-46. Sydney, Australia. Doi:10.1111/j.1442-9993.2001.01070.pp.x

Arora, R., Goel, S., & Mittal, R. K. (2017). Supporting collaborative software development over GitHub. Software: Practice and Experience, 47(10), 1393-1416. Retrieved from: https://onlinelibrary.wiley.com. Doi: 10.1002/spe.2468.

Atoum, I., & Bong, C. H. (2015). Measuring Software Quality in Use: State-of-the-Art and Research Challenges. Software Quality Professional, 17(2), 4. Retrieved from: http://asq.org/pub/sqp/past/vol20_issue1/index.html

Badashian, A. S., & Stroulia, E. (2016). Measuring user influence in GitHub: the million follower fallacy. Paper presented at the Proceedings of the 3rd International Workshop on CrowdSourcing in Software Engineering. Beijing, China.

Bahamdain, S. S. (2015). Open Source Software (OSS) Quality Assurance: A Survey Paper. Procedia Computer Science, 56, 459-464. Doi:10.1016/j.procs.2015.07.236.

Barnett, J. G., Gathuru, C. K., Soldano, L. S., & McIntosh, S. (2016). The relationship between commit message detail and defect proneness in Java projects on GitHub. Paper presented at the Proceedings of the 38th International Conference on Software Engineering. ACM .New York, USA.

Baudry, B., & Monperrus, M. (2012). Towards ecology inspired software engineering. arXiv preprint arXiv:1205.1102. Retrieved from: https://arxiv.org/abs/1205.1102

Baumgartner, H., & Homburg, C. (1996). Applications of structural equation modeling in marketing and consumer research: A review. International journal of Research in Marketing, 13(2), 139-161. Retrieved from: https://www.sciencedirect.com/journal

Bavota, G., Gethers, M., Oliveto, R., Poshyvanyk, D., & Lucia, A. d. (2014). Improving software modularization via automated analysis of latent topics and dependencies. ACM Transactions on Software Engineering and Methodology (TOSEM), 23(1), 1-33. Doi:10.1145/2559935

Bello-Orgaz, G., Jung, J. J., & Camacho, D. (2016). Social big data: Recent achievements and new challenges. Information Fusion, 28, 45-59. Doi: 10.1016/j.inffus.2015.08.005

Benjamini, Y., & Braun, H. (2002). John Tukey's contributions to multiple comparisons. ETS Research Report Series, 2002(2). Doi: 10.1002/j.2333-8504.2002.tb01891.x

Biazzini, M., & Baudry, B. (2014). May the fork be with you: novel metrics to analyze collaboration on GitHub. Paper presented at the 36th International Conference on Software Engineering. Hyderabad, India.

Bissyandé, T. F., Lo, D., Jiang, L., Réveillere, L., Klein, J., & Le Traon, Y. (2013a). Got issues? who cares about it? a large-scale investigation of issue trackers from GitHub. Paper presented at the IEEE 24th International Symposium on on Software Reliability Engineering (ISSRE). Retrieved from: https://ieeexplore.ieee.org.

Bissyandé, T. F., Thung, F., Lo, D., Jiang, L., & Réveillere, L. (2013b). Popularity, interoperability, and impact of programming languages in 100,000 open source projects. Paper presented at the Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual.

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

Boin, A., & Fishbacher-Smith, D. (2011). The importance of failure theories in assessing crisis management: The Columbia space shuttle disaster revisited. Policy and Society, 30(2), 77-87. Doi:10.1016/j.polsoc.2011.03.003

Borges, H., Hora, A., & Valente, M. T. (2016a). Predicting the popularity of GitHub repositories. In Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering (p. 9). ACM.

Borges, H., Hora, A., & Valente, M. T. (2016b). Understanding the factors that impact the popularity of GitHub repositories. Paper presented at the Software Maintenance and Evolution (ICSME). Raleigh, North Carolina, United States.

Borges, H., Valente, M. T., Hora, A., & Coelho, J. (2015). On the popularity of GitHub applications: A preliminary note. arXiv preprint arXiv:1507.00604.

Bose, L., & Thakur, S. (2013). Introducing Agile into a Non-Agile Project Analysis Of Agile Methodology With Its Issues And Challenges. International Journal of Advanced Research in Computer Science, 4(2), 305-311.

Brunetti, G., Feld, T., & Heuser, L. (2014). Future Business Software Current Trends in Business Software Development (Vol. 1;2014;). S.l.: Springer International Publishing.

Campos, L. M., & Scherson, I. D. (2000). Rate of change load balancing in distributed and parallel systems. Parallel Computing, 26(9), 1213-1230.

Capra, E., Francalanci, C., Merlo, F., & Rossi-Lamastra, C. (2011). Firms' involvement in Open Source projects: A trade-off between software structural quality and popularity. Journal of Systems and Software, 84(1), 144-161. Doi:10.1016/j.jss.2010.09.004

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., . . . Riddell, A. (2017). Stan: A probabilistic programming language. Journal of statistical software, 76(1).

Casalnuovo, C., Vasilescu, B., Devanbu, P., & Filkov, V. (2015, August). Developer onboarding in GitHub: the role of prior social links and language experience. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (pp. 817-828). ACM.

Chatziasimidis, F., & Stamelos, I. (2015). Data collection and analysis of GitHub repositories and users. Paper presented at the 6th International Conference on Information, Intelligence, Systems and Applications (IISA). Corfu, Greece. Doi: 10.1109/IISA.2015.7388026.

Chen, F., Li, L., Jiang, J., & Zhang, L. (2014). Predicting the number of forks for open source software project. Paper presented at Proceedings of the 2014 3rd International Workshop on Evidential Assessment of Software Technologies. Pages 40-47 Nanjing, China. Doi:10.1145/2627508.2627515.

Cheng, C., Li, B., Li, Z.-Y., Zhao, Y.-Q., & Liao, F.-L. (2017). Developer Role Evolution in Open Source Software Ecosystem: An Explanatory Study on GNOME. Journal of Computer Science and Technology, 32(2), 396-414. Doi:10.1007/s11390-017-1728-9

Cheng, C., Li, Z., Li, B., & Liang, P. (2018). Automatic Detection of Public Development Projects in Large Open Source Ecosystems: An Exploratory Study on GitHub. arXiv preprint arXiv:1803.03175.

Cho, T. (2014). Improved techniques for automatic chord recognition from music audio signals. New York University. New York, USA. ISBN: 978-1-3037-6422-6.

Choi, N., & Yi, K. (2015). Raising the general public's awareness and adoption of open source software through social Q&A interactions. Online Information Review, 39(1), 119-139. Doi:10.1108/oir-06-2014-0139

Chou, S.-W., & He, M.Y. (2011). The factors that affect the performance of open source software development - the perspective of social capital and expertise integration. Information Systems Journal, 21(2), 195-219. Doi:10.1111/j.1365-2575.2009.00347.x

Christopher V, M. L.V., Asquez, G. B., & Massimiliano Di Penta, D. G. A. D. P. (2015). license Usage and Changes: A Large-Scale Study of Java Projects on GitHub. Paper presented at: Proceedings of the 38th International Conference on software engineering companion. Austin, USA. Doi:10.1145/2889160.2889259. Paderborn, Germany.

Coelho, J., & Valente, M. T. (2017). Why modern open source projects fail. Paper presented at: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. 186-196. Doi:10.1145/3106237.3106246.

Cortés-Coy, L. F., Linares-Vásquez, M., Aponte, J., & Poshyvanyk, D. (2014). On automatically generating commit messages via summarization of source code changes. Paper presented at: 14th International Working Conference on Source Code Analysis and Manipulation (SCAM), IEEE, 2014. Doi: 10.1109/SCAM.2014.14

Cosentino, V., Luis, J., & Cabot, J. (2016). Findings from GitHub: methods, datasets and limitations. In Proceedings of the 13th International Conference on Mining Software Repositories (pp. 137-141). ACM. Doi:10.1145/2901739.2901776

Cosentino, V., Izquierdo, J. L. C., & Cabot, J. (2017). A Systematic Mapping Study of Software Development with GitHub. IEEE Access, 5, 7173-7192. Doi: 10.1109/ACCESS.2017.2682323

Cunningham, E. (2008). A practical guide to structural equation modelling using Amos. Deakin University, Melbourne, Australia. Retrieved from: https://blogs.deakin.edu.au

Da Silva, A. C. B. G., de Figueiredo Carneiro, G., de Paula, A. C. M., Monteiro, M. P., & e Abreu, F. B. (2016). Agility and Quality Attributes in Open Source Software Projects Release Practices. Paper presented at the 10th International Conference on the Quality of Information and Communications Technology (QUATIC) (2016). Lisbon, Portugal.

Dabbish, L., Stuart, C., Tsay, J., & Herbsleb, J. (2012). Social coding in GitHub: transparency and collaboration in an open software repository. Paper presented

at the Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work. Doi:10.1145/2145204.2145396

Dias, L. F., Steinmacher, I., Pinto, G., da Costa, D. A., & Gerosa, M. (2016). How does the shift to GitHub impact project collaboration? Paper presented at 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), (pp. 473-477). Doi: 10.1109/ICSME.2016.78

Diaz, A., Merino, P., & Rivas, F. J. (2010). Mobile application profiling for connected mobile devices. IEEE Pervasive Computing, 9(1), 54-61. Doi:10.1109/MPRV.2009.63.

Dingsøyr, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. The Journal of Systems and Software, 85(6), 1213-1221. Doi:10.1016/j.jss.2012.02.033

Driscoll, W. C. (1996). Robustness of the ANOVA and Tukey-Kramer statistical tests. Computers & industrial engineering, 31(1-2), 265-268. Retrieved from https://doi.org/10.1016/0360-8352(96)00127-1

Fangohr, H. (2004). A comparison of C, MATLAB, and Python as teaching languages in engineering. In International Conference on Computational Science (pp. 1210-1217). Springer, Berlin, Heidelberg. Retrieved from https://doi.org/10.1007/978-3-540-25944-2_157.

Franco-Bedoya, O., Ameller, D., Costal, D., & Franch, X. (2017). Open source software ecosystems: A Systematic mapping. Information and Software Technology, 91, 160-185. Elsevier B.V. Retrieved from https://doi.org/10.1016/j.infsof.2017.07.007

Gandomani, T. J., Zulzalil, H., Ghani, A. A. A., & Sultan, A. B. M. (2012)., A systematic literature review on relationship between agile SD and open source SD, International review on computers and software (IRECOS), Vol. 7, Issue 4, pp. 1602-1607. Retrieved from: https://arxiv.org/abs/1302.2748

Gousios, G. (2013). The GHTorent dataset and tool suite. MSR '13 Proceedings of the 10th working conference on mining software repositories (pp. 233-236). IEEE Press San Francisco, CA, USA.

Gousios, G., & Spinellis, D. (2012). GHTorrent: GitHub's data from a firehose. In 2012 9th IEEE Working Conference on Mining Software Repositories (MSR), (pp. 12-21). Doi: 10.1109/MSR.2012.6224294

Gousios, G., & Spinellis, D. (2017). Mining software engineering data from GitHub. In Software Engineering Companion (ICSE-C), IEEE/ACM 39th International Conference (pp. 501-502). IEEE. Doi: 10.1109/ICSE-C.2017.164

Gousios, G., Vasilescu, B., Serebrenik, A., & Zaidman, A. (2014). Lean GHTorrent: GitHub data on demand. In Proceedings of the 11th working conference on mining software repositories (pp. 384-387). ACM. Doi:10.1145/2597073.2597126

Goyal, R., Ferreira, G., Kästner, C., & Herbsleb, J. (2018). Identifying unusual commits on GitHub. Journal of Software: Evolution and Process, 30(1), e1893. Doi:10.1002/smr.1893

Grapentine, T. (2000). Path analysis vs. structural equation modeling. Marketing research, 12(3), 12. Chicago, USA. Retrieved from: https://search-proquest-com.elibrary.

Gunal, V. (2012). Agile Software Development Approaches and Their History. Enterprise Software Engineering. Retrieved from: https://sewiki.iai.uni-bonn.de

Haigh, T. (2011). The history of information technology. Annual Review of Information Science and Technology, 45(1), 431-487. Retrieved from https://doi.org/10.1002/aris.2011.1440450116

Hair, J. F., Black, W. C., Babin, B. J., Anderson, R. E., & Tatham, R. L. (1998). Multivariate data analysis (Vol. 5): Prentice hall Upper Saddle River, NJ.

Hamilton, John R., and Tee, Singwhat (2013) *Understanding social network site consumer engagements.* In: Proceedings of the 24th Australasian Conference on Information Systems. From: 24th Australasian Conference on Information Systems, 4-6 December 2013, Melbourne, VIC, Australia.

Hamilton, J. R., & Tee, S. (2015). Expectations-to-value: connecting customers with business offerings. International Journal of Internet Marketing and Advertising, 9(2), 121-140. Retrieved from: https://doi.org/10.1504/IJIMA. 2015.070716

Härdle, W. K., & Borke, L. (2017). GitHub API based QuantNet Mining infrastructure in R. Retrieved from: https://papers.ssrn.com/sol3/papers.cfm?abstract_id= 2927901

Hata, H., Todo, T., Onoue et al. (2015). Characteristics of sustainable OSS projects: A theoretical and empirical study. In Proceedings of the Eighth International Workshop on Cooperative and Human Aspects of Software Engineering (pp. 15-21). IEEE Press. Florence, Italy.

Hebig, R., Quang, T. H., Chaudron, M. R., Robles, G., & Fernandez, M. A. (2016). The quest for open source projects that use UML: mining GitHub. In Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (pp. 173-183). ACM. Saint-malo, France. doi:10.1145/2976767.2976778

Heirich, M. (1964). The use of time in the study of social change. American Sociological Review. Vol. 29, No. 3, pp. 386-397. doi: 10.2307/2091482

Henderson, S. (2009). How do people manage their documents? An empirical investigation into personal document management practices among knowledge workers, Thesis (PhD)--University of Auckland, 2009. Auckland. Retrieved from: http://hdl.handle.net/2292/5230

Hertel, G., Niedner, S., & Herrmann, S. (2003). Motivation of software developers in Open Source projects: An Internet-based survey of contributors to the Linux kernel. Research Policy, 32(7), 1159-1177. Doi:10.1016/s0048-7333(03)00047-7

Higham, D. J., & Higham, N. J. (2016). MATLAB guide (Vol. 150): SIAM: Society for Industrial and Applied Mathematics; 2 Edition. ISBN-13: 978-0898715781

Hu, Y., Zhang, J., Bai, X., Yu, S., & Yang, Z. (2016). Influence analysis of GitHub repositories. Springerplus, 5(1), 1-19. Doi:10.1186/s40064-016-2897-7

Jain, A., & Gupta, M. (2017). Evolution and Adoption of programming languages. Evolution, 5(1). nternational Journal of Modern Computer Science (IJMCS). Retrieved from: http://www.ijmcs.info/current_issue/IJMCS170233.pdf

Jarczyk, O., Gruszka, B., Jaroszewicz, S., Bukowski, L., & Wierzbicki, A. (2014). GitHub projects. quality analysis of open-source software. In International Conference on Social Informatics (pp. 80-94). Lecture Notes in Computer Science, vol 8851. Springer, Cham. Doi: 10.1007/978-3-319-13734-6_6

Jiang, J., Lo, D., He, J., Xia, X., Kochhar, P. S., & Zhang, L. (2017). Why and how developers fork what from whom in GitHub. Empirical Software Engineering, 22(1), 547-578. Doi:10.1007/s10664-016-9436-6

Jibaja, I., Jensen, P., Hu, N., Haghighat, M. R., McCutchan, J., Gohman, D. & McKinley, K. S. (2015, October). Vector Parallelism in JavaScript: Language and compiler support for SIMD. In 2015 International Conference on Parallel Architecture and Compilation (PACT) (pp. 407-418). IEEE. San Francisco, CA, USA.

Jones, C. (2014). Software Industry Goals for the Years 2014 through 2018. Journal of Cost Analysis and Parametrics, 7(1), 41-47. Doi:10.1080/1941658X.2014.890493

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., & Damian, D. (2016). An in-depth study of the promises and perils of mining GitHub. Empirical Software Engineering, 21(5), 2035-2071. Springer US. Retrieved from: https://doi.org/10.1007/s10664-015-9393-5.

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., & Damian, D. (2014). The promises and perils of mining GitHub. Paper presented at the Proceedings of the 11th working conference on mining software repositories. Doi:10.1145/2597073.2597074.

Kilamo, T., Hammouda, I., Mikkonen, T., & Aaltonen, T. (2012). From proprietary to open source - Growing an open source ecosystem. Journal of Systems and Software, 85(7), 1467-1478. Doi:10.1016/j.jss.2011.06.071.

King, G. (1986). How not to lie with statistics: Avoiding common mistakes in quantitative political science. American Journal of Political Science, Vol. 30,

No. 3 (Aug., 1986), pp. 666-687. Midwest Political Science Association. Doi: 10.2307/2111095.

Kline, R. B. (2015). Principles and practice of structural equation modeling: Fourth Edition. Guilford publications. Paperback, 534 Pages, Published 2015 by The Guilford Press, New York. US. ISBN-13: 978-1-4625-2334-4.

Kumar, K., & Dahiya, S. (2017). Programming Languages: A Survey. Change, International Journal on Recent and Innovation Trends in Computing and Communicati 5(5). IJRITCC. Haryana, India. Retrieved from: http://www.ijritcc.org.

Lakhani, K. R., & Von Hippel, E. (2003). How open source software works: "free" user-to-user assistance. Research Policy, 32(6), 923-943. Doi:10.1016/s0048-7333(02)00095-1

Lanubile, F., Ebert, C., Prikladnicki, R., & Vizcaíno, A. (2010). Collaboration tools for global software engineering. IEEE Software, 27(2). Doi:10.1109/MS. 2010. 39

Lee, M. J., Ferwerda, B., Choi, J., Hahn, J., Moon, J. Y., & Kim, J. (2013). GitHub developers use rockstars to overcome overflow of news. In CHI'13 Extended Abstracts on Human Factors in Computing Systems (pp. 133-138). ACM. Paris, France. Doi:10.1145/2468356.2468381

Li, L., Goethals, F., Baesens, B., & Snoeck, M. (2017). Predicting software revision outcomes on GitHub using structural holes theory. Computer Networks, Volume 114, pp 114-124. Retrieved from: https://doi.org/ 10.1016/j.comnet.2016.08.024

Liao, Z., He, D., Chen, Z., Fan, X., Zhang, Y., & Liu, S. (2018). Exploring the Characteristics of Issue-Related Behaviors in GitHub Using Visualization Techniques. IEEE Access, 6, 24003-24015. Doi: 10.1109/ACCESS.2018.2810295

Lima, A., Rossi, L., & Musolesi, M. (2014). Coding Together at Scale: GitHub as a Collaborative Social Network. Proceedings of the Eighth International AAAI Conference on Weblogs and Social Media (ICWSM 2014). Retrieved from: http://www.aaai.org/ocs/index.php/ICWSM/ICWSM14/paper/download/8112/8130

Luo, Z., Mao, X., & Li, A. (2015, August). An exploratory research of GitHub based on graph model. In Frontier of Computer Science and Technology (FCST), 2015 Ninth International Conference on (pp. 96-103). IEEE. Doi: 10.1109/FCST.2015.45

Ma, W., Chen, L., Zhou, Y., & Xu, B. (2016, September). What Are the Dominant Projects in the GitHub Python Ecosystem? In Trustworthy Systems and their Applications (TSA), 2016 Third International Conference on (pp. 87-95). IEEE. Doi: 10.1109/TSA.2016.23

Manikas, K., & Hansen, K. M. (2013). Software ecosystems – A systematic literature review. Journal of Systems and Software, 86(5), 1294-1306. Doi:10.1016/j.jss.2012.12.026

Markovtsev, V., & Kant, E. (2017). Topic modeling of public repositories at scale using names in source code. arXiv preprint arXiv:1704.00135. Retrieved from: https://arxiv.org/abs/1704.00135

Marlow, J., Dabbish, L., & Herbsleb, J. (2013, February). Impression formation in online peer production: activity traces and personal profiles in GitHub. In Proceedings of the 2013 conference on Computer supported cooperative work (pp. 117-128). ACM. San Antonio, Texas, USA. Doi:10.1145/2441776.2441792.

Matragkas, N., Williams, J. R., Kolovos, D. S., & Paige, R. F. (2014, May). Analysing the 'biodiversity' of open source ecosystems: the GitHub case. In Proceedings of the 11th Working Conference on Mining Software Repositories (pp. 356-359). ACM. Hyderabad, India. Doi:10.1145/2597073.2597119.

Melosan, I. (2014). The application of analysis of variance (ANOVA) to different experimental results of c45 medium-carbon steel., vol. 66, iss. 2, (2014): 30-35. Bucharast, Romania.

Mens, T., Claes, M., Grosjean, P., & Serebrenik, A. (2014). Studying evolving software ecosystems based on ecological models. In Evolving Software Systems (pp. 297-326). Springer, Berlin, Heidelberg.

Mileva, Y. M. (2012). Mining the evolution of software component usage. PhD Dissertation, Saarland University, 1-104. Retrieved from: https://publikationen.sulb.uni-saarland.de/handle/20.500.11880/26438

Munaiah, N., Kroh, S., Cabrey, C., & Nagappan, M. (2017). Curating GitHub for engineered software projects. Empirical Software Engineering, 22(6), 3219-3253. Doi:10.1007/s10664-017-9512-6

Muthén, L. K., & Muthén, B. O. (2002). How to Use a Monte Carlo Study to Decide on Sample Size and Determine Power. Structural Equation Modeling: A Multidisciplinary Journal, 9(4), 599-620. Doi:10.1207/S15328007SEM0904_8

Nixon, R. (2014). Learning PHP, MySQL, JavaScript, CSS & HTML5: A Step-by-Step Guide to Creating Dynamic Websites: O'Reilly Media, Inc. CA. USA

Noone, M., & Mooney, A. (2017). Visual and Textual Programming Languages: A Systematic Review of the Literature. Journal of Computers in Education. arXiv preprint arXiv:1710.01547. Doi:10.1007/s40692-018-0101-5

Nurdiani, I., Börstler, J., & Fricker, S. A. (2016). The impacts of agile and lean practices on project constraints: A tertiary study. Journal of Systems and Software, 119, 162-183. Retrieved from: https://doi.org/10.1016/j.jss.2016.06.043

Olson, D. L., & Rosacker, K. (2012). Crowdsourcing and open source software participation. Service Business, 7(4), 499-511. Doi:10.1007/s11628-012-0176-4

Onoue, S., Hata, H., & Matsumoto, K.-I. (2013). A study of the characteristics of developers' activities in GitHub. 20th Asia-Pacific Software Engineering Conference (APSEC). Doi: 10.1109/APSEC.2013.104.

Orii, N. (2012). Collaborative Topic Modeling for Recommending GitHub Repositories. School of Computer Science, Carnegie Mellon University, Pittsburgh, USA. Retrieved from: http://www.cs.cmu.edu/~norii/pub/ GitHub-ctr.pdf

Padhye, R., Mani, S., & Sinha, V. S. (2014, May). A study of external community contribution to open-source projects on GitHub. In Proceedings of the 11th Working Conference on Mining Software Repositories (pp. 332-335). ACM. Hyderabad, India. Doi:10.1145/2597073.2597113

Papadopoulos, G. (2015). Moving from Traditional to Agile Software Development Methodologies Also on Large, Distributed Projects. Procedia - Social and Behavioral Sciences, 175, 455-463. Doi:10.1016/j.sbspro.2015.01.1223

Papaioannou, T., Wield, D., & Chataway, J. (2009). Knowledge ecologies and ecosystems. Environmental and planning c: government and policy, 27(2), 319-339. Doi:10.1068/c0832

Peterson (2013). The GitHub open source development process. Retrived from: http://kevinp.me/GitHub-process-research/GitHub-process-research.pdf

Pianosi, F., Sarrazin, F., & Wagener, T. (2015). A MATLAB toolbox for global sensitivity analysis. Environmental Modelling & Software, 70, 80-85. Retrieved from: https://doi.org/10.1016/j.envsoft.2015.04.009

Qassimi, N. A., & Rusu, L. (2015). IT Governance in a Public Organization in a Developing Country: A Case Study of a Governmental Organization. Conference on Enterprise Information Systems 2015 (CENTERIS 2015). Vol. 64, p. 450-456. Elsevier. Retrieved from: https://www.sciencedirect.com/science/article/pii/S1877050915026769

Randell, B. (1996). The 1968/69 NATO software engineering reports. History of Software Engineering, 37. A conference sponsored by the NATO Science Committee. Garmisch, Germany. Retrieved from: http://homepages.cs.ncl.ac.uk/brian.randell/NATO/NATOReports/

Ray, B., Posnett, D., Filkov, V., & Devanbu, P. (2014). A large-scale study of programming languages and code quality in GitHub. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (pp. 155-165). ACM. Hong Kong, China. Doi:10.1145/2635868.2635922

Ribeiro, A., & da Silva, A. R. (2012, September). Survey on cross-platforms and languages for mobile apps. In Quality of Information and Communications Technology (QUATIC), 2012 Eighth International Conference on the (pp. 255-260). IEEE. Lisbon, Portugal. Doi: 10.1109/QUATIC.2012.56

Robinson, W. N., & Deng, T. (2015). Data mining behavioral transitions in open source repositories. In System Sciences (HICSS), 2015 48th Hawaii International Conference on (pp. 5280-5289). IEEE. Kauai, HI, USA. Doi: 10.1109/HICSS.2015.622.

Sarka, P., & Ipsen, C. (2017). Knowledge sharing via social media in software development: a systematic literature review. Knowledge Management Research & Practice, 15(4), 594-609. Retrieved from: https://www.tandfonline.com/doi/abs/10.1057/s41275-017-0075-5.

Schmidt, D. C., Stal, M., Rohnert, H., & Buschmann, F. (2013). Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects (Vol. 2): John Wiley & Sons. University of California, Irvine, USA. ISBN: 978-1-118-72517-7.

Shah, H., Allard, R. D., Enberg, R., Krishnan, G., Williams, P., & Nadkarni, P. M. (2012). Requirements for guidelines systems: Implementation challenges and lessons from existing software-engineering efforts. BMC Medical Informatics and Decision Making, 12(1), 16-16. Doi:10.1186/1472-6947-12-16

Sharma, A., Thung, F., Kochhar, P. S., Sulistya, A., & Lo, D. (2017, June). Cataloging GitHub repositories. In Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering (pp. 314-319). ACM. Karlskrona, Sweden. doi:10.1145/3084226.3084287

Sheoran, J., Blincoe, K., Kalliamvakou, E., Damian, D., & Ell, J. (2014, May). Understanding watchers on GitHub. In Proceedings of the 11th Working Conference on Mining Software Repositories (pp. 336-339). ACM. Hyderabad, India. Doi:10.1145/2597073.2597114

Siau, K., & Tian, Y. (2013). Open Source Software Development Process Model: A Grounded Theory Approach. Journal of Global Information Management (JGIM), 21(4), 103-120. Doi: 10.4018/jgim.2013100106

Singer, L., Figueira Filho, F., & Storey, M. A. (2014, May). Software engineering at the speed of light: how developers stay current using twitter. In Proceedings of the 36th International Conference on Software Engineering (pp. 211-221). ACM. Hyderabad, India. Doi: 10.1145/2568225.2568305

Soll, M., & Vosgerau, M. (2017, September). ClassifyHub: An Algorithm to Classify GitHub Repositories. In Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz) (pp. 373-379). Springer, Cham. Retrieved from: https://doi.org/10.1007/978-3-319-67190-1_34

Song, C., Wang, T., Yin, G., Zhang, X., & Yang, C. (2016). A Novel Open Source Software Ecosystem: From a Graphic Point of View and Its Application. 2016, 71-74. Doi:10.18293/seke2016-123. Retrieved from http://ksiresearchorg.ipage.com/seke/seke16paper/seke16paper_123.pdf

Squire, M. (2014, January). Forge++: The changing landscape of FLOSS development. In System Sciences (HICSS), 2014 47th Hawaii International Conference on (pp. 3266-3275). IEEE. Waikoloa, HI, USA. Doi: 10.1109/HICSS.2014.405

Squire, M. (2017, May). Considering the Use of Walled Gardens for FLOSS Project Communication. In IFIP International Conference on Open Source Systems (pp. 3-13). Springer, Cham. Retrieved from: https://link.springer.com/Chapter/10.1007/978-3-319-57735-7_1.

Syeed, M. M., Hansen, K. M., Hammouda, I., & Manikas, K. (2014, August). Socio-technical congruence in the ruby ecosystem. In Proceedings of The International Symposium on Open Collaboration (p. 2). ACM. Berlin, Germany. Doi: 10.1145/2641580.2641586

Tachizawa, T., & Pozo, H. (2012). Management model for the development of Software applied to business sustainability in the context of global climate changes. Journal of Information Systems & Technology Management, 9(1), 39. Sao Paulo, Brazil doi:10.4301/S1807-17752012000100003

Tsay, J., Dabbish, L., & Herbsleb, J. (2014a). Influence of social and technical factors for evaluating contribution in GitHub. In Proceedings of the 36th international conference on Software engineering (pp. 356-366). ACM. Hyderabad, India. Doi: 10.1145/2568225.2568315.

Tsay, J., Dabbish, L., & Herbsleb, J. (2014b). Let's talk about it: evaluating contributions through discussion in GitHub. In Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering (pp. 144-154). ACM. Hong Kong, China. Doi:10.1145/2635868.2635882

Van der Maas, J. C. (2016). Evolution of Collaboration in Open. Master's thesis. Department of Information and Computer Science, Utrecht University. Utrecht, Netherlands. Retrieved from: https://www.uu.nl/en/education/archive-masters-thesis

Vasilescu, B., Blincoe, K., Xuan, Q., Casalnuovo, C., Damian, D., Devanbu, P., & Filkov, V. (2016, May). The sky is not the limit: multitasking across GitHub projects. In Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on (pp. 994-1005). IEEE. Austin, TX, USA. Doi: 10.1145/2884781.2884875

Vasilescu, B., Yu, Y., Wang, H., Devanbu, P., & Filkov, V. (2015, August). Quality and productivity outcomes relating to continuous integration in GitHub. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software

Engineering (pp. 805-816). ACM. Bergamo, Italy. Doi:10.1145/2786805.2786850

Weber, S., & Luo, J. (2014, December). What makes an open source code popular on git hub? In Data Mining Workshop (ICDMW), 2014 IEEE International Conference on (pp. 851-855). IEEE. Shenzhen, China. Doi: 10.1109/ICDMW.2014.55

West, J., & Gallagher, S. (2006). Challenges of open innovation: the paradox of firm investment in open-source software. R&D Management, 36(3), 319-331. Wiley Digital archive. https://doi.org/10.1111/j.1467-9310.2006.00436.x

Williams, L. (2012). What agile teams think of agile principles. Communications of the ACM, 55(4), 71-76. New York, NY, USA. Doi:10.1145/2133806.2133823

Wu, Y., Kropczynski, J., Shih, P. C., & Carroll, J. M. (2014, February). Exploring the ecosystem of software developers on GitHub and other platforms. In Proceedings of the companion publication of the 17th ACM conference on Computer supported cooperative work & social computing (pp. 265-268). ACM. Baltimore, Maryland, USA. Doi:10.1145/2556420.2556483

Xavier, J., Macedo, A., & de Almeida Maia, M. (2014). Understanding the popularity of reporters and assignees in the GitHub. In SEKE (pp. 484-489). Retrieved from: https://scholar.google.com/citations?user=8Haa9vQAAAAJ&hl=it

Ye, Y., & Kishida, K. (2003). Toward an understanding of the motivation Open Source Software developers. Paper presented at the Proceedings of the 25th international conference on software engineering. Portland, USA. Doi:10.1109/ICSE.2003.1201182

Yu, Y., Wang, H., Yin, G., & Ling, C. X. (2014a). Reviewer recommender of pull-requests in GitHub. In Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on (pp. 609-612). IEEE. Victoria, BC, Canada. Doi: 10.1109/ICSME.2014.107

Yu, Y., Wang, H., Yin, G., & Wang, T. (2016). Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment? Information and Software Technology, 74, 204-218. Retrieved from: https://doi.org/10.1016/j.infsof.2016.01.004

Yu, Y., Yin, G., Wang, H., & Wang, T. (2014b). Exploring the patterns of social behavior in GitHub. In Proceedings of the 1st international workshop on crowd-based software development methods and technologies (pp. 31-36). ACM. Hong Kong, China. Doi:10.1145/2666539.2666571

Zakiah, A., & Fauzan, M. N. (2016, April). Collaborative Learning Model of Software Engineering using GitHub for informatics student. In Cyber and IT Service Management, International Conference on (pp. 1-5). IEEE. Bandung, Indonesia. Doi: 10.1109/CITSM.2016.7577521

Zhu, J., Zhou, M., & Mockus, A. (2014). The relationship between folder use and the number of forks: A case study on GitHub repositories. ESEM, Torino, Italy. Retrieved from: http://mockiene.com/papers/folder-short.pdf

APPENDICES

Appendix A: Standardized Total Effects

Table A-1: Standardized Total Effects for 195 JavaScript language repos

JavaScript 195 Repos	Watches	Stars	Forks	Issues	Pulls	Contributors	Releases
Issues	0.45	-	-	-	-	-	-
Pulls	0.31	-	0.14	0.69	-	-	-
Contributors	0.01	0.40	0.11	0.52	0.76	-	-
Releases	0.20	0.41	-0.24	0.55	0.19	0.25	-
Commits	0.32	0.06	0.03	0.74	0.50	0.04	0.16

Table A-2: Standardized Total Effects for 195 Python language repos

Python 195 Repos	Watches	Stars	Forks	Issues	Pulls	Contributors	Releases
Issues	-0.35	0.28	0.44	-	-	-	-
Pulls	-0.19	0.15	0.24	0.55	-	-	-
Contributors	-0.19	0.15	0.43	0.54	0.56	-	-
Releases	-0.22	0.18	0.24	0.62	-0.10	-0.17	-
Commits	-0.19	0.15	0.23	0.54	0.80	-0.03	0.15

Table A-3: Standard total Effects for 170 Java Language Repos

JAVA 170 Repos	Watches	Stars	Forks	Issues	Pulls	Contributors	Releases
Issues	-	0.37	-	-	-	-	-
Releases	-	0.26	-0.16	0.69	0.16	0.24	-
Pulls	-	0.17	-	0.45	-	-	-
Contributors	-	0.2	-	0.54	0.68	-	-
Commits	0.31	-0.1	-0.02	0.72	0.41	0.3	0.15

Table A-4: Standardized Total Effects for C# language repos

199 repos for C# language	Watches	Stars	Forks	Issues	Pulls	Contributors	Releases
Issues	-	0.28	0.23	-	-	-	-
Pulls	-	0.20	0.17	0.71	-	-	-
Contributors	-0.40	0.47	0.42	0.45	0.63	-	-
Release	-	0.06	0.05	0.20	0.28	-	-
Commits	-0.07	0.20	0.17	0.49	0.69	0.19	0.30

Table A-5: Standardized Total Effects for C++ language repos

196 Repos for C++ language	Watches	Stars	Forks	Issues	Pulls	Contributors	Releases
Issues	-	0.37	-	-	-	-	-
Pulls	-	0.03	0.41	0.52	-	-	-
contributor	-0.33	0.38	0.57	0.31	0.60	-	-
release	-	0.07	0.1	0.29	0.25	-	-
commits	-0.07	0.16	0.19	0.55	0.70	0.20	0.21

Table A-6: Standardized Total Effects for CSS language repos

199 Repos for CSS language	Watches	Stars	Forks	Issues	Pulls	Contributors	Releases
Issues	0.36	-	-	-	-	-	-
Pulls	0.15	-0.38	0.4	0.43	-	-	-
Contributors	0.08	-0.37	0.39	0.23	0.97	-	-
Release	0.2	0.07	-0.07	0.56	-0.17	-	-
Commits	0.21	-0.27	0.17	0.58	0.7	0.31	0.11

Table A-7: Standardized Total Effects for PHP language repos

197 Repos for PHP language	Watches	Stars	Forks	Issues	Pulls	Contributors	Releases
Issues	-	-	0.49	-	-	-	-
Pulls	-	-	0.47	0.7	-	-	-
contributors	-0.39	0.27	0.76	0.48	0.7	-	-
Release	-0.07	0.28	0.04	0.62	0.13	0.19	-
Commits	-0.01	0.04	0.38	0.84	0.63	0.03	0.14

Table A-8: Standardized Total Effects for Ruby language repos

165 Repos for Ruby	Watches	Stars	Forks	Issues	Pulls	Contributors	Releases
Issues	-0.2	0.69	-	-	-	-	-
Pulls	-0.11	0.39	0.13	0.56	-	-	-
Releases	-0.1	0.35	-	0.51	-	-	-
Contributors	-0.11	0.51	0.08	0.53	0.61	-	-
Commits	0.07	0.15	0.04	0.78	0.3	0.49	0.22

LIST OF PUBLICATIONS

Alshomali, Mohammad Azeez, Holdsworth, Jason, and Hamilton, John (2016) *Identifying ways of supporting software development in the open source community.* In: Proceedings of the 20th International Conference on ISO & TQM. From: 20 ICIT: 20th International Conference on ISO & TQM, 26-28 September 2016, Al Buraimi, Oman.

Alshomali, Mohammad Azeez, Holdsworth, Jason, and Hamilton, John R. (2017) *A preliminary exploration of the GitHub ecosystem: how to find important repositories.* In: Proceedings of ISCA 2017, pp. 346-352. From: ISCA 2017: 1st Iraqi Scholars Conference in Australasia, 5-6 December 2017, Melbourne, VIC, Australia.

Hamilton, John R., Holdsworth, Jason, Tee, SingWhat, and **Alshomali, Mohammad Azeez** (2017) *Analysing big data projects using GitHub and JavaScript repositories.* In: Proceedings of the 17th International Conference on Electronic Business, pp. 47-52. From: ICEB 2017: 17th International Conference on Electronic Business, 4-8 December 2017, Dubai, United Arab Emirates.

Alshomali, Mohammad Azeez, Hamilton, John R., Holdsworth, Jason, and Tee, SingWhat (2017) *GitHub: factors influencing project activity levels.* In: Proceedings of the 17th International Conference on Electronic Business, pp. 116-124. From: ICEB 2017: 17th International Conference on Electronic Business, 4-8 December 2017, Dubai, United Arab Emirates.