MEASURES OF SOFTWARE DEVELOPMENT LEADING TO APPLICATIONS ENHANCEMENTS

A THESIS SUBMITTED TO AUCKLAND UNIVERSITY OF TECHNOLOGY
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF COMPUTER AND INFORMATION SCIENCES

Supervisor

Prof. A Litchfield

28th of October 2023

By

Tasleem Faizal Hussein

School of Engineering, Computer and Mathematical Sciences

Abstract

This study analyses measures and patterns from data drawn from application communications and updates from development teams, to assess the value of data-driven prioritisation of software enhancements, and to develop a framework to facilitate decision-making. The data source used is GitHub and datasets include software user interactions that influence the software development lifecycle of projects such as feature requests, bug reports and issues logged and their resolutions. The dataset also includes software developer activities against a software projects such as code commits, cloning of repositories, peer review activities relating to code changes and releases of changes. Multivariate time series data analysis methods are applied to identify which factors influence software changes. The results are used in this study to create models and frameworks for decision-making based on available data.

Businesses have encountered rapid changes in customer demands for content and quality of software delivered in short time frames resulting in software development teams adopting ways of working and technologies to release software as Minimal Viable Products with continual delivery of enhancements. This presents an opportunity to use metrics to prioritise product enhancements to continuously deliver valuable software. However, there exist opportunities for changes to be made that do not provide value and this study seeks to establish value can be ascribed.

Contents

List of Tables

List of Figures

Attestation of Authorship

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the qualification of any other degree or diploma of a university or other institution of higher learning.

Signature of student

Acknowledgements

I would like to thank my supervisor Dr. Alan Litchfield for all his patience and advice during the writing of this thesis. I would also like to express my gratitude to my wife, Shazmeen Hussein, for her continual encouragement and support. Without the support of Shazmeen this journey would not have been possible.

Chapter 1

Introduction

1.1 Introduction

This study investigates the activities of software development to product insights which influence software development activities relating to delivering value to software users and organisations that publish software. The study investigates how the value of software enhancements is identified and implemented and the impact of the releases of the software changes on both users and publishers of software applications.

Software is used daily by users across the world in multiple aspects of daily activities from navigation through to entertainment and revenue generation activities in organisations. Users of software derive value from functionality software provides and publishers of software try to maintain and enhance the value of software through software delivery activities. Determining current value software provides and identifying future value software can provide can assist software publishers in identifying software changes which may introduce new functionality or enhance existing functionality to allow users in getting more out of the applications they interact with.

Value of software is determined in multiple ways and can inform the prioritisation of software development efforts resulting in code changes to applications. Examples

of value is usability, quality and functionality. Software delivery resources are often constrained, so determining the value being delivered needs to be justifiable compared to the resources used in delivering it. Value also needs to be identified and delivered promptly to introduce positive changes to software users quickly.

With businesses experiencing increasing customer demands for organisations to deliver high-quality content and features in short timeframes. This leads to a dynamic environment where software must evolve rapidly and released value to its users frequently (Bucena & Kirikowa, 2019). Therefore timely and accurate identification of value and the ability to prioritise value delivery can minimise existing software user dissatisfaction or result in an increase of new software users. If software changes can be released quickly into the hands of customers, they are able to realise value such as increased adoption, customer satisfaction or get customer feedback faster, resulting in quick evolution of software to meet customer needs and wants (Smith, Villalba, Irvine, Stanke & Harvey, 2021). Therefore timely and accurate identification and prioritisation of value can also help allocate resources to impactful tasks resulting in efficient use of limited resources.

Identification and delivery of value in software delivery can also help organisations meet business objectives. Delivery of value through software enhancements can lead to and increase market share, reputation benefits and differentiation with competitors (Binzagr, Labbaci & Medjahed, 2019). This can assist with the sustainability of software and the ability to increase the resources to software delivery through increased adoption and revenue resulting in an increase in an organisations ability to delivery value (Wlodarski, Poniszewska-Maranda & Falleri, 2022).

Through the work presented in this study, an evidence based approach to the identification of value resulting in software enhancements is investigated. To achieve this, value of software is defined and an evidence based model to predict software enhancements is created and presented.

1.2 Problem Statement

Software publishers are encountering increasing levels of market competition. As a result of this competition, customers have alternate options to software titles if they experience dissatisfaction in using applications (Wang, Axtell & Loerch, 2017). As such organisations that publish software need to understand the impact of software releases to the quality of their products and resulting value delivered to customers. Put simply, software releases need to ensure value delivered, protected and improved to help increase customer satisfaction (Krishnan, 1993).

Determining value resulting in software changes is difficult to quantify as there are many definitions and measurements of value. In addition to the difficulty in quantifying and defining the value of software changes, the methods in which information is used to determine changes and their value can differ between software delivery teams. This can result in value not being identified and therefore missed, value being identified too late or value being prioritised incorrectly leading to a mis-allocation of resources.

Software delivery teams work within the constraints of the resources available to them. With the adoption of agile practices, software developers are required to contribute more than coding tasks to projects. Examples of these tasks can be assisting in defining quality assurance tests, application behaviour from a user perspective, documentation and project management and communications tasks. Although these tasks may positively influence the quality of software, resulting in an increase to user software adoption rates or customer satisfaction (Kalliamvakou, Gousios, Spinellis & Nancy, 2009), these tasks take time away from making code changes resulting in a delivery slowdown.

Along with software delivery resource constraints, software projects generally have fixed time and monetary resources. With these limited resources, organisations need to focus on software delivery of features that provide the most impact to customers

and retain value. The constraints, as mentioned earlier (resources) and objectives (quality, speed and customer satisfaction), result in the need for software delivery teams to identify value in order to allocate time and costs at each stage of the Software Development Life Cycle (SDLC) (Yang & Zhang, 2008).

1.3 Thesis structure

This research uses the Design Science Research (DSR) methodology and time series statistical analysis to investigate how software delivery teams can use evidence-based methods to identify and deliver meaningful value to software users. This requires defining value in terms of software delivery, collecting and analysing data for open-source software projects, and conducting analysis on the data collected to produce an artefact that software delivery teams can use in addressing the research problem.

To help achieve the aforementioned outcomes, this thesis is structured as follows. Chapter 2 contains a literature review performed to define software delivery, value in the context of software and the drivers of software enhancements. The literature review helps produce a structured set of research questions and can be considered the problem definition phase of DSR.

Chapter 3 describes the method used to carry out the study to address the problem identified in the previous chapter. This forms the objective definition and design and development phases of DSR where problems are linked to objectives and an approach to solve the problem and achieve outcomes is specified.

Chapter 4 details the findings derived from executing the design artefact specified in the previous chapter. This consists of results obtained from statistical analysis of data collected from GitHub and the creation of an artefact that can be used to solve the research problem, which for the purposes of this study is an SVARIMA forecasting model.

Chapter 5 discusses the findings detailed in the previous chapter. This is done by articulating the implications of the analysis carried out in Chapter 4 on the research questions posed at the end of Chapter 2. Any further insights derived from analysis are also discussed in this chapter along with the impact of limitations on this study.

Chapter 6 provides a summary of the study carried out in this thesis. The findings of using the research methods chosen for this study, along with the outcomes of the analysis in answering the research questions posed, are summarised in this chapter. Challenges encountered in this study and potential future research are also discussed here.

Chapter 2

Literature Review

2.1 Introduction

This review seeks to identify what evidence based approaches are used in the software delivery process to identify and prioritise enhancements. To do this, this literature review attempts to answer the following research questions:

1. In what ways can a software development team use a data-driven approach to improve or enhance software products?

2. Of the measures identified, what is the level of acceptance and use for each?

The literature review finds that a relationship between the value and quality of software exists from the perspective of software users. The quality of software operating in a technically and functionally correct manner for its users is what influences the value provided by the software. Frameworks that attempt to measure the quality of software exist, measuring the technical correctness of software and the functionality it provides to solve software user problems.

The review has not found frameworks that are used to provide an evidence based

approach to identifying value that is used by software delivery teams publishing applications for users. This indicates that the processes used to prioritise software enhancements made to software can vary between publishers, and the value of enhancements may be misidentified.

In this chapter, the literature reviewed to understand the definition of software in Section 2.2 and value Section 2.3 is described. The chapter also investigates how software enhancements are managed by software delivery teams Section 2.4 and software users attempt to use evidence based approaches to evaluate the quality of software to determine its value Section 2.5. Finally, the research questions synthesised from the literature review are defined in Section 2.7.

2.2 Definition of Software

This section presents a definition of software and methods used to specify the use of artefacts by users.

Software, also referred to as applications, packages or binaries, can be defined as "a set of instructions that directs a processor (hardware) to perform actions" (Joca, 2016). Software can be published as standalone applications accessed directly by human users or as a component that is accessed indirectly by users through other software applications (Xu & Brinkkemper, 2005). Organisations that create software, referred to as Software Vendors (Maalej, Nayebi & Ruhe, 2019), make software available to users through the following commercial models:

Commercial software a construct where software source code making up the hardware instructions are kept confidential but the functionality of the software is released to users. Software can be purchased for use by organisations termed Software Vendors or used based on license restrictions at no charge. Purchased

software is typically licensed before being able to be used and so requires permission from the vendors. Licenses may allow the use of software in perpetuity or on a subscription basis that is renewed at predetermined intervals (Ferrante, 2006).

Open source software a construct allowing software and its underlying source code to be publicly available and editable. Open source software typically has license agreements that governs commercial distribution and use aimed to ensure fair use and distribution. Examples of this may be license restrictions prohibiting organisations selling the open source software without permission (Xu & Brinkkemper, 2005) or requirements that changes made to underlying code are fed back to the open source community.

The use of software can manifest itself daily in the form of mobile phones, cars, work or entertainment (Ebert, 2014). This results in software vendors having access to a large pool of existing or potential users who can help grow revenue along with competitors that can disrupt market share. As such the value that software delivers to users is an area of importance for vendors requiring identification of functionality users want or levels of quality their products supply.

To assist with retaining existing and gaining new customers and competing with other software vendors who operate in the same industry vertical, organisations have adopted a software product management view to their software delivery processes. To achieve the previously mentioned outcomes, product management teams are tasked with identifying and delivering new software or changes to existing software that are valuable to their users. The value of software is discussed further in the next section.

2.3 Value of software

The following section discusses the concept of software value and how it is currently defined and informs the software delivery efforts of organisations that publish software.

Organisations that publish software attempt to create products or artefacts that satisfies the requirements of key stakeholders therefore providing value (Ramzan, Jaffar & Shahid, 2009). Software publishers have placed importance on identifying, prioritising and delivering value to users (Z. S. Li et al., 2024) of software which has resulted in the introduction of:

- Product Management which seeks to identify customer needs and translate them into software enhancements discussed in Section 2.4.
- Frameworks such as Agile ways of working (Beck et al., 2001) which emphasises the prioritisation of activities that result in customer satisfaction through delivery of value (Alahyari, Svensson & Gorschek, 2017).
- Lean software development principles that emphasises a focus on ensuring activities and processes result in adding value to the organisation publishing software or customers using software (Poppendieck, 2007).

Identification and delivery of value impacts customer satisfaction and the market relevance of software. As such the definition of value, how it is understood by stakeholders and the return that the realisation of value will deliver is an area of importance to software publishers (Alahyari et al., 2017).

2.3.1 Definition of value

When value is not explicitly defined, organisations may focus time and resources on activities that don't result in realisation of value and achievement of goals. As such contributors to the software management and delivery process need to agree on what value their software provides and make decisions that focus actions on maximising the delivery identified value (Biffl, Aurum, Boehm, Erdogmus & Grünbacher, 2006). To

help with the identification of value, software publishers need to agree on a definition of value.

This literature review identifies the following commonly used definitions of value (Biffl et al., 2006):

- Exchanging a product or service for another based on the quantity or quality of items involved in the transaction.
- The principle of group behavior where members of groups have a strong commitment that provides benchmarks used to evaluate actions, objectives and outcomes.
- Spiritual or aesthetic qualities, practical usefulness, material or monetary worth that is gained through the exchange of objects.

2.3.2 Definition of software value

The definitions in Section 2.3.1 can be considered social and economic definitions of value and can be used to inform the definition of software value as "The degree of fulfillment of stakeholder's requirements in order of their priority while maintaining the agreed upon commitments and constraints of quality" (Ramzan et al., 2009).

The definition of software value has three concepts for discussion (Ramzan et al., 2009). This literature review identifies stakeholders that may be software users, delivery teams or software vendors. The concept of priority needs to inform the software delivery efforts resulting in efforts being applied to delivering value that's of importance to stakeholders discussed in Section 2.4. Finally there is the concept of quality which may be of importance to multiple stakeholders such as customers, software delivery teams, software management teams and organisations that publish software discussed in Section 2.5 and Section 2.6.

2.4 Software Product Management

In Section 2.3.2 value of software is identified, and the need for the software delivery process needing to prioritise value to stakeholders is discussed. This section of the literature review discusses the Software Product Management Model and how it attempts to structure the identification and prioritisation of value.

A software product management model is an approach to help focus software delivery efforts on customer outcomes (Miguel, Mauricio & Rodríguez, 2014). Software vendors identify that software functionality and quality are important to customers (Lin & Huang, 2009) however, this cannot be viewed with a technology-only lens (Ebert, 2014). As such software vendors apply focus to continually identifying and validating customer needs their applications attempt to meet to ensure software engineering efforts are applied most effectively (Olsson & Bosch, 2014).

To evaluate customer needs, prioritise software development efforts and maximise customer value delivery, organisations adopt software product management as a framework (Fricker, 2012) that attempts to:

- Identify and understand market segments software to deliver value in and the market size.
- Identify and understand customers within market segments and what value they attempt to derive from software features.
- Identify and understand emerging or embedded competitors within identified markets and their value proposition to customers.
- Prioritise software delivery efforts to focus on maximise customer value delivered.
- Help ensure the quality of the product is at a level that customers can derive value and have a reliable experience.

2.5 Quality

Quality is identified as an indicator of value to stakeholders that publish and use software, as discussed in Section 2.3.2. In this section, the quality of software applications is defined. Quality can be defined as "The degree to which a system, component or process meets the specified requirements" and "The degree to which a system, component or process meets the needs or expectations of a user" (IEEE, 1983).

The definition of quality has a strong relationship to the outcomes desired by a software product management framework as mentioned in Section 2.4. The purpose of software product management can be stated as the want to identify the value customers desire or need and quality is defined as the realisation of those requirements. If the functionality that software delivers is aligned to the value customers expect, then it is reasonable to assume adoption of software trends upwards.

Software users also measure the quality of software (Samoladas, Gousios, Spinellis & Stamelos, 2008). Although there is a commonality in the measures vendors and users use to assess the quality of software, the methods used and data available to use a data-driven process can differ between the two parties. Quality models have been developed to assist users and vendors in measuring the value of software consistently and are further discussed in the next section.

2.6 Quality Models

There are multiple models used to measure the quality of software that product management teams and customers use. In this section, metrics and models which help in measuring outcomes that impacts a customers satisfaction and ensure delivery of value are discussed. Although there is no single framework that is agreed upon, the Factor/Criteria/Metrics methodology (McCall, Richards & Walters, 1977) for measuring

software quality is identified as the basis of modern software quality assessment models (Lin & Huang, 2009).

The Factor/Criteria/Metrics methodology considers quality being measured using a process that determines factors that contribute to software quality, criteria that identify the factors and definition of metrics that can measure or with each factor to determine a rating for it.

Software product managers (Hsi & Potts, 2000), developers (Niedermaier, Koetter, Freymann & Wagner, 2019) and users (Binzagr et al., 2019) are three groups of stakeholders identified as users of quality models to assess software viability.

2.6.1 The McCall Quality Model

This review has the McCall Quality model as an established software quality assessment model (Rawashdeh & Matalkah, 2006). This section defines the McCall Quality Model and discusses how it considers the evaluation of software to assess quality, beneficial concepts this model introduces and some model limitations.

The McCall Quality Model (McCall et al., 1977) is continuously modified to adopt a wider range of characteristics as the industry definition of software quality evolves and both software publishers and users seek a standardised approach to software quality evaluation (Ramzan et al., 2009). The McCall Quality Model (Miguel et al., 2014) suggests quality assessment criteria can be grouped into two interrelated perspectives containing evaluation criteria. The perspectives are the assessment of external quality can be measured by customers and internal quality measured by programmers. This introduces the concept of quality needing to be assessed by both the publishers of software and its users.

McCall considers the relationship between quality criteria and metrics, for example, the reliability of software is a criterion of assessment that can be measured by accuracy,

consistency and error tolerance, however the metrics required a binary Yes or No answer. The quality model does not focus on the functionality of an application a user would want to assess. This leads to the use of this model in assessing the suitability of software being focused on technical requirements not value delivered in providing a solution for a need (Miguel et al., 2014).

2.6.2 The ISO 9126 Quality Model

The ISO 9126 Quality Model is an extension of the McCall Quality Model described in Section 2.6.1. In this section, the ISO 9126 Quality model is defined and discussions on how it articulates the evaluation of software, beneficial concepts and limitations of this model.

The ISO 9126 Quality model, (Miguel et al., 2014), states that software quality evaluation needs to consider the quality of a software product and the effect of software products on the users. Similar to the McCall model, ISO 9126 recommends software needs to be assessed from the perspectives of internal quality and external quality. ISO 9126 extends the McCall model by considering a third perspective of quality in use criteria that assesses the effectiveness of the products (Miguel et al., 2014). Quality in use criteria refer to the effectiveness of software, how it impacts user productivity, the effectiveness of the application in solving customer needs and the security of the application and resulting customer satisfaction. The Quality in use perspective can be equated to measuring the value that software delivers to a customer.

ISO 9126 considers the concept of value as needing assessment when considering the quality of software. This extends the McCall model, which focuses on the technical evaluation of software. ISO 9126 does not however specify metrics that need to be measured for the criteria specified and so requires evaluators to specify them, resulting in no industry stand metrics being defined.

When assigning stakeholders likely to use the ISO 9126 evaluation model, it is found that end users, product managers and software developers would use the model to evaluate an application. With no quality-in-use metrics defined, there may be misalignment in what end users expect from software and what the software delivery teams may identify as features that deliver value (Rawashdeh & Matalkah, 2006).

2.7 Research Questions and Hypotheses

This section articulates the research questions derived from the literature review executed in this chapter. In Section 2.4 the process of continually identifying and validating customer needs is discussed, but no evidence-based frameworks for software enhancement feature selection is found. This leads to the formation of RQ1 and RQ3. The literature review also identifies the concept of value to both software publishers and users. Value can influence the adoption of software, as discussed in Section 2.3, and so RQ1 and RQ2 are noted for further investigation in this study.

2.7.1 Research questions

RQ1 How can a software development team select software updates that adds value with confidence?

RQ2 What are the factors that effect or determine value?

RQ3 How can an evidence-based approach be applied to software enhancements?

2.8 Conclusion

This review investigates how the value of software applications is defined and assessed by stakeholders creating software and consuming it. The review identifies the link

between value and quality and frameworks used to measure quality from the perspective of software producers and users. Findings indicate that the quality of software from an end user goes beyond technical considerations as end users need to evaluate the value proposition of software in how it solves problems they are encountering or the value they wish to realise.

Findings indicate that organisations producing software want to deliver value to customers to increase adoption rates through delivering customer satisfaction. However, in the reviewed literature, well-defined characteristics are found in quality models checking for technical considerations, but there are no industry-defined model that specifies metrics that evaluate the value software delivers to end users.

Chapter 3 articulates the research methods used to answer the research questions synthesised in the literature review carried out in this chapter.

Chapter 3

Method

3.1 Introduction

Chapter 2 presents a literature review that defines software, the value of software and ways in which software delivery teams identify and prioritise software enhancements. The literature review shows that processes related to the identification and prioritisation of software enhancements use disparate methods, and a major objective of software delivery teams is to increase the value of software to its users. The literature review also demonstrates that the perception of value can vary between publishers and users of software applications resulting in misidentified leading to questionable value being released. As a result the research questions posed in Section 2.7.1 are formed with the objective of determining how evidence can be used in the identification and prioritisation of software enhancements.

In this chapter presents a research methodology and describes the research plan that is applied to answer the research question in Section 2.7.1:

RQ1 How can a software development team select software updates that adds value with confidence?

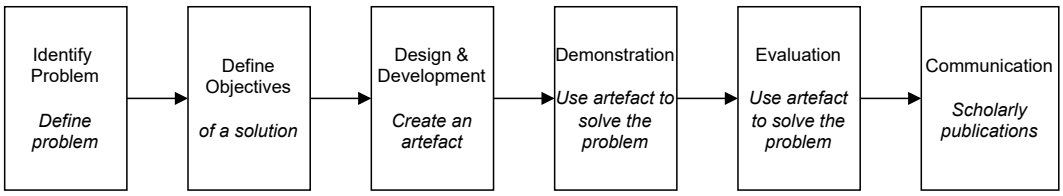

Figure 3.1: DSR Empirical process

RQ2 What are the factors that effect or determine value?

RQ3 How can an evidence based approach be applied to software enhancements?

3.2 Design Science Research Methodology

This section describes the DSR framework. DSR is described as the empirical investigation of the performance of physical and abstract entities of the world, their attributes, behaviours and interactions. DSR defines entities created by humans as artefacts where artefacts provide functionality to help solve problems (Sordi, 2021). Examples of artefacts may be organisations, tools or frameworks. DSR, as in Figure 3.1, requires the empirical investigation of artefacts to be conducted in a context. In Information Systems the context can be categorised as the design, development, maintenance or use of software applications (Wieringa, 2014).

When using DSR it is important to identify the type of problems being investigated. Problems can be categorised as design problems or knowledge questions and the categorisation informs the research methods to be applied. Design problems are defined here as a problem relating to the design or re-design of artefacts to better solve a problem relating to a goal (Peffers, Tuunanen, Rothenberger & Chatterjee, 2007). Design problems are treated by following the design cycle which proposes a problem to be investigate, treatment designed and validation of the treatment to be carried out. Knowledge questions are defined as questions posed to get knowledge about the world as it is. Knowledge questions are answered by following the empirical cycle and answers

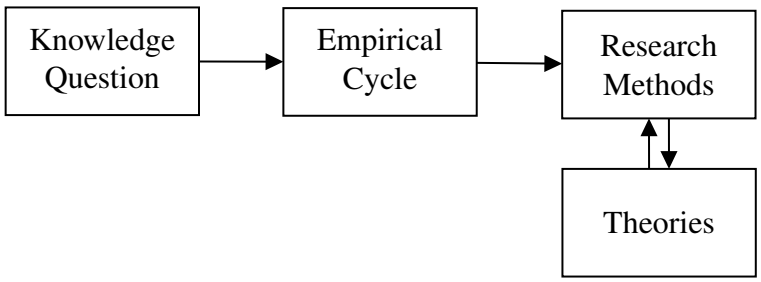

Figure 3.2: DSR Framework

should be evaluated to test the truthfulness of answers and researchers need to assume that answers are fallible (Wieringa, 2014). Once problems are categorised into a design or empirical cycle, the research design steps can be applied and theories produced that can assist in refining iterations of the research design.

To answer the research questions in Section 2.7.1, this thesis seeks to identify the causes of software enhancements through the analysis of the SDLC of software project artefacts in the context of software enhancements. This study proposes the events resulting in software enhancements can be classified as interactions between artefacts and context. As such, we seek to answer a question and gain more knowledge about software enhancements. Due to this rationale, this thesis classifies the research problem as a knowledge question and uses the DSR empirical cycle research methodologies highlighted in Figure 3.2.

Figure 3.2 describes the DSR framework and is adopted from Wieringa (2014, Fig 1) to remove the research methods describing the steps required to solve a design problem as this thesis seeks to answer a knowledge question.

3.3 Research Study Design

This section details the steps of the DSR process model and how it is applied to the research design in this study, the statistical analysis methods used, data sources used, and case selection criteria at each step of the DSR process. DSR defines an

Empirical study as "a rational way to answer scientific knowledge questions" (Wieringa, 2014). The research methods used to carry out an empirical study using DSR, as shown in Figure 3.1 is adopted from Peffers et al. (2007, Figure 1 DSRM Process Model). Figure 3.1 illustrates a process of identifying a problem, defining research objectives and solutions, creating an artefact to solve the research problem, validating the artefact through demonstrations and evaluation and communicating the outcomes of the research.

This research uses DSR as it is a process designed to solve real-world problems which fits into the domain of this research subject area which is software development. This is further detailed in Section 3.4. The DSR process specifies the creation of artefacts to use in solving problems allowing the creation of frameworks to use in identifying valuable software enhancements. This aligns with the research goals specified in Section 3.5. This study produces such an artefact detailed in Chapter 4. The demonstration and evaluation steps in the DSR process encourage rigorous evidence-based approaches, which require demonstration and evaluation of artefacts produced. This has a strong alignment with the research problem and objectives this thesis identifies, which centre on evidence-based decision-making. The demonstration and evaluation of an artefact produced in Chapter 4 is carried out in Chapter 5. This study produces a time-series predictive model that is used to identify, and forecast, user feature enhancement requests or bugs. Such a model is used to examine the impact of software releases on users by measuring and forecasting their feature and bug fix requests.

This chapter is structured to align the research method used with the DSR Empirical process as listed below:

Identify Problem Section 3.4 defines the research problem this study looks to solve and how it is identified.

Define Objectives The definition of a problem in the Identify Problem process allows

us to specify the research objectives and hypothesis for this study which are detailed in Section 3.5.

Design and Development With the problem, objectives and hypothesis specified, a solution to solve the problem and reach the research objectives in Section 3.6 is created. This section details the selection of a data source, data collection methods, case selection and evaluation criteria and statistical methods used to analyse data. These processes combine to create an artefact which is evidence-based and can be demonstrated and evaluated.

Demonstration and evaluation Section 3.7 contains details about the statistical methods used to demonstrate and evaluate artefacts produced by this research.

Communication The outputs produced by this research are the communication mechanism for the DSR process followed. The Research Method, Analysis and Discussion chapters of this study forms the publication of the artefact, findings and insights.

3.4 Problem Definition

In Chapter 2 a literature review is carried out to understand the SDLC, how the value of software can be measured using evidence-based approaches and how software delivery teams manage the SDLC of their applications to deliver value. The results help formulate the research questions presented in Section 2.7.1. In this section, the findings of the literature review and how they lead to the definition of the research problem to be addressed in this study are discussed.

The literature review identifies that software can be published to users on the basis that it is paid for or permanently free and that software is used both at points in time during the day or continually by its users. The literature review also determines that the software market was contested and open for disruption, where new titles could displace

existing published software titles (Binzagr et al., 2019). As such, the concept of value is identified as being a key driver for both software publishers and users. A perception of the value of a software title could lead to an increase or attrition of user adoption or retention. The findings above also leads to the emergence of the concept of time to market where the speed at which you release enhancements to software can have an effect on your users.

The literature identifies that the value of software can be assessed using frameworks that define quality in the categories of functional and non-functional characteristics of software (Ramzan et al., 2009). Functional characteristics of software categorise the value software provides to end users to meet their needs within the problem domain the software serves and non-functional characteristics define the performance, security and reliability of software. The concept of value is difficult to quantify, and existing quality frameworks, which are attempts to categorise value, suggest that actors using quality assessment frameworks place their own weightings against categories of quality. As such the concept of value and quality is influenced by various factors such as functional and non-function aspects of software previously mentioned.

The Software Product Management process and framework is identified as a method used by software delivery teams in identifying software enhancements that can improve existing or create new functional and non-functional characteristics of software (Biffl et al., 2006). Frameworks and processes residing within the Software Product Management domain attempt to prioritise the efforts of software delivery teams in improving software to get greater adoption or retain existing users. These efforts are constrained by the existing knowledge or resources within software delivery teams, and so the concepts of resource constraints and prioritisation emerge. The literature review does not find frameworks or processes involving the use of evidence-based approaches in selecting software enhancements.

The findings above lead to the challenge of effective software enhancement selection.

The continuous enhancement of software within the resource constraints of software delivery teams and the competitive market in which software publishers operate necessitates frequent delivery of meaningful value to software users. This literature review has found no existing frameworks that assist with the identification of value, and a lack of systematic approaches to the identification of value. As a result, the risk of misidentification of software enhancements exists and could result in the determination of value of software delivery efforts being incorrect.

Due to the findings mentioned above, this study aims to address these challenges by investigating the SDLC for open-source projects hosted in GitHub to identify factors that determine the value of software enhancements, how an evidence-based approach can be used to identify software enhancements and ways in which software delivery teams can select software enhancements with confidence that value is delivered to users as described in the research questions specified in Section 2.7.1.

3.5 Objectives Definition

In this section, the research objectives of this study and the hypothesis this study seek to test are defined.

3.5.1 Research Objectives

Research objectives help define aims of a study, guide the research study design and assist researchers to focus on important outcomes (Farrugia, Petrisor, Farrokhyar & Bhandari, 2010). Research objectives need to consider time and resource constraints a researcher has to work within and so should be specific and achievable and be inferred from the research problem and associated research questions identified (Peffers et al., 2007).

The objectives of this research are as follows:

O1 Produce an evidence-based framework that can be used by software delivery teams to identify software enhancements

O2 Measure how software enhancements responding to software user requests impacts the adoption of applications

3.5.2 Research Hypothesis

In this section the research hypothesis for this study are defined. A hypothesis is a research instrument used to decide if sample cases analysed are able to support generalisations. Hypothesis can be defined as a single or set of propositions formulated as an explanation of phenomena to guide investigations. It is crucial for a research hypothesis to be tested for validity in a critical manner and are able to be objectively verified (Kothari, 2004). Figure 3.3 provides a mapping between the research question and hypothesis.

The hypothesis for this research are as follows:

H1 factors that identify software enhancements can be identified using evidence based methods

H2 factors identifying software enhancements can be used to create an evidence based framework to identify software enhancements

H3 using an evidence based framework to select software enhancements increases the value of software enhancements to software users

H4 using an evidence based framework to identify software enhances is beneficial to software delivery teams

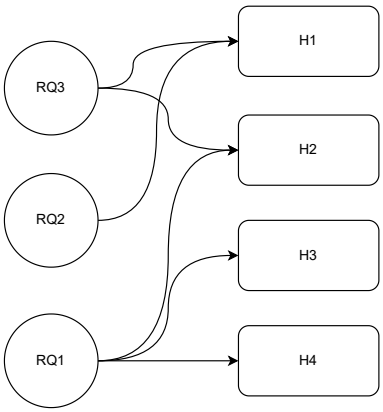

Figure 3.3: Research Question to Hypothesis Mapping

3.6 Design and Development

In this section, the research design and artefact development steps which are a part of the DSR Empirical process illustrated in Figure 3.1 are defined. DSR defines design and development as the process of creating an artefact that can help solve the research problem(s) identified and meet the research objectives (Peffers et al., 2007). The process of creating an artefact requires a structured approach using various research methods that are described in this section.

The design and development process carries out the following steps. First data sources for analysis are selected for analysis. Section 3.6.1 details the data source selected for this study and the reasoning behind the choice. With a data source selected, the methods used to collect data from the data source and prepare it for analysis are specified.

Section 3.6.2 specifies the process of collecting, transforming and validating data and making it available for analysis. Cases are then selected for analysis, and the case selection criteria are detailed in Section 3.6.3. Cases selected require validation to ensure they are aligned with the research questions and objectives. The statistical methods used to validate and analyse selected cases and the reasoning behind them are described in Section 3.6.4. Section 3.6.4 also describes the statistical methods used to

create an artefact that assists in answering the research questions.

3.6.1 Data Source

The research objectives identified in Section 3.5.1 require the identification of evidence resulting in software enhancements and so no intervention is required to answer the research question. As such this research uses a case study approach to measure phenomena in the object of study. The cases studied are chosen from publicly available Opensource projects selected from GitHub and are listed in Table 4.1. GitHub is an online code collaboration and repository hosting site (Gousios & Spinellis, 2017) that manages software projects used by more than eighty million users globally and was acquired as a subsidiary of Microsoft in 2018 (Microsoft, 2018). GitHub is used as it is the most widely accepted and used project collaboration and source code management tool that is publicly accepted resulting in a good representation of software project populations (Wessel, Vargovich, Gerosa & Treude, 2023).

Interactions between interested actors and GitHub repositories are recorded as GitHub events. Examples of GitHub events are code commits (`Pull` Requests) and issues logged (`Issues` Events). GitHub events are time-stamped and versioned data points which are used in studies to explore the popularity of open source projects, software development social dynamics and other phenomena (Varuna & Mohan, 2019). There are currently over 45 events on GitHub used by teams collaborating on software development projects. This study uses GitHub events to answer the research questions in Section 2.7.1. The details of data collection are further articulated in Section 3.6.2.

Due to the large number of projects on GitHub, projects selected for study in this study need to be representative of the population of projects in GitHub. This study acknowledges that projects used will only be representative of Opensource software and may not result in generalisations that could be applied to closed-sourced commercial

software. GitHub projects are, from this point on, referred to as cases.

Although GitHub allows data collection through a set of Application Programmer Interfaces (APIs), the interfaces do not provide an easy method for collecting large sets of metadata for all repositories hosted on GitHub. Also, the resulting data sets are not structured for easy analysis. Therefore this research uses GH Archive as a source for GitHub metadata. GH Archive is a research tool that collects GitHub events daily and collates them in a structure suitable for analysis and reporting purposes (Hagiwara & Mita, 2019). GH Archive is often used for academic research purposes as it provides trends over time and non-aggregated data. GH Archive also allows researchers to avoid limitations placed on GitHub APIs, such as resource consumption limits and the number of daily operations (AlMarzouq, AlZaidan & AlDallal, 2020).

3.6.2 Data Collection

In this section the process of pulling data from GH Archive and the data pipeline process used to make it available for analysis is detailed. The solution illustrated in Figure 3.4 is created to pull data from GitHub via GH Archive into an environment for further processing and analysis in the scope of answering the research questions previously mentioned in Section 2.7.1.

To allow analysis, the collected GitHub events data require preprocessing steps to ensure data is valid and compatible with analytical tools. The data preprocessing pipeline illustrated in Figure 3.4 encompasses three steps listed below:

The details of the tools used in Figure 3.4 are detailed in Section 3.8.

1. **Data Retrieval:** Using the `wget` command in a Linux operating system, data from GH Archive APIs is pulled into a data staging storage area for further processing. Data from the first of January 2018 through to the 18th of September 2022 is downloaded. This is noted as **1** in Figure 3.4.

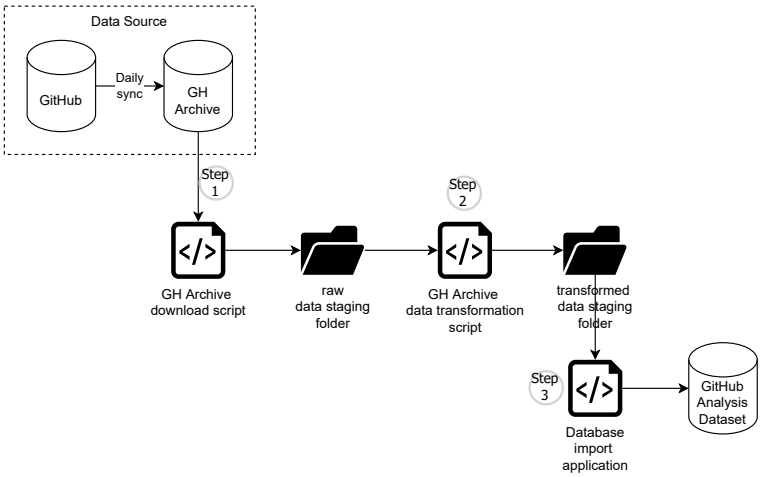

Figure 3.4: Import Solution High Level Design

2. **Data Transformation:** The data retrieved from GH Archive is compressed and in JSON format, which is transformed into CSV format for ease of import into a relational database. This conversion simplified the subsequent data import and validation process allowing for a simplified mapping of source data fields to their corresponding database tables. This is noted as **2** in Figure 3.4.

3. **Data Import and Validation:** The transformed CSV files are imported into a relational database, allowing for efficient querying. During the import process, data validation is carried out to identify issues relating to the integrity and consistency of the imported dataset. This includes handling mapping errors, invalid characters, missing values and format discrepancies. This process is noted as **2** in Figure 3.4.

3.6.3 Sample Selection

In this section, the process of identifying projects hosted on GitHub as cases for the purposes of statistical analysis is described. GitHub is an online code collaboration and repository hosting site (Gousios & Spinellis, 2017) that manages software projects

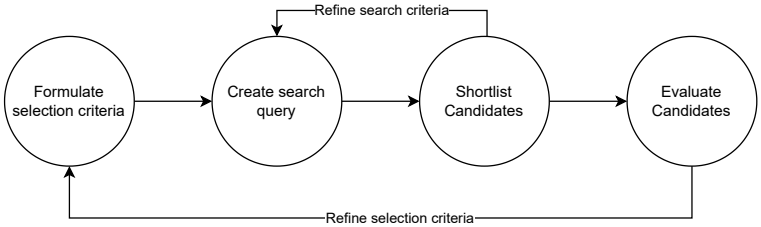

Figure 3.5: GitHub repository candidate assessment process

used by more than eighty million users globally and was acquired as a subsidiary of Microsoft in 2018 (Microsoft, 2018). Since its' founding in 2008 GitHub has grown to have more than 28 million public repositories publicly available and can range from server-side software that does not require end-user interaction to operate through to user-focused software that can run on mobile or windows operating systems (Zhang, Chen, Luo & Li, 2019).

This study proposes to also use open source projects hosted in GitHub to answer the questions posed in Section 2.7.1. Many open-source projects use GitHub for source control, documentation, collaboration, logging, and categorising issues (Bissyande et al., 2013) with data made available through GitHub or GH Archive APIs. This study uses the process illustrated in Figure 3.5 to find projects in GitHub and evaluate the candidate repositories for use as cases in this study. Due to the large number of projects hosted on GitHub, case selection of projects needs an iterative approach as inappropriate results may need to be filtered out and the search run again.

The high level process in Figure 3.5 is further detailed in the following sections of this chapter and is summarised below. Further details of the process can be in Section 3.6.3.

Formulate selection criteria is the process of specifying selection criteria that can be applied to projects hosted in GitHub to help answer this studies research question.

Create query and Search GitHub is a combination of translating the search criteria into a query that can be executed using the GitHub Search tool and ranking the

results in order to pick five candidate repositories.

Refine search criteria is the process of search criteria refinement that can be executed if search results return unsuitable candidates. Search criteria refinements can be the adjustment of search query filters but needs to ensure that search query refinements maintain alignment to the project selection criteria.

Evaluate Candidates is the process of gathering additional metrics to further evaluate short listed candidates for use in this study. This process requires creating of an evaluation criteria variables database, collection of metric and non metric data and application of analysis of the variables to find three candidate projects.

Refine selection criteria is the process of reviewing and refining the selection criteria if the evaluation of candidate repositories results in unsuitable projects being returned. Examples of this are a lack of developer or user community activity against a repository.

Formulate Selection Criteria

For the purposes of the research question this study looks to answer, the selection criteria for projects to analyse rules out applications that are not user-facing. As such, the search attribute `Topic` excludes frameworks, runtimes and documentation. Runtimes and frameworks are tools used by technologists that provide pre-defined libraries of code that can contain reusable logic. As such they are not directly user-facing and also used by technologists who typically favour software development leading to a limitation in the user base. Documentation can be related to the use of software or APIs, runtimes and frameworks. These are at times user-facing but typically do not get feature requests as the end user community can self-edit them.

This study only considers projects that have at least three years of data to ensure a sufficient volume of information for analysis existed. To ensure projects have a large support group and a variety of users the search for repositories that hosted code for

text editor workloads. Using text editors as cases for statistical analysis is based on the reasoning below:

Technology proliferation Text editors can run on multiple operating systems, avoiding a bias of case selection by software (operating systems) or hardware (device) technologies.

Wide Usage Text editors are widely used by professionals such as professionals, academics and casual technology users. This allows feedback from a wide variety of application user demographics.

Frequent updates Text editors are typically used continually throughout the year. This leads to regular requests for enhancements for performance or functional improvements. Text editors are also impacted by the frequent release of libraries they use to write or render code and so this is another influencing factor in ensuring cases selected have a large number of events that can be analysed to answer the research question. Finally, text editors are typically customizable to serve end-user use cases and so can provide evidence of user satisfaction with releases.

Impact on end-user productivity Functionality of text editors significantly impacts user productivity and efficiency. Changes released can potentially improve or degrade end-user experience which could be captured as issues logged by users.

With the selection criteria for cases defined, candidate GitHub project repositories are searched using the selection criteria specified in Table 3.1.

Table 3.1: Project search criteria

Search Attribute	Value	Search Operator
Topic	editor	Include

Continued on next page

Table 3.1 – continued from previous page

Search Attribute	Value	Search Operator
	framework	Exclude
	runtime	Exclude
	documentation	Exclude
Created	2019-01-01	Less than

Search GitHub for Repositories and Shortlist Candidates

The selection criteria defined in Table 3.1 are translated into the following GitHub search syntax `topic:editor NOT runtime NOT documentation NOT framework create` and entered into the GitHub search tool located at `https://github.com/search`. Search results are sorted by `Most Stars` in descending order and the top 5 results are listed in Table 4.1.

Evaluate Candidates

GitHub projects captured in Table 3.1 returned by the search parameters specified in Section 3.6.3 are further investigated to ensure they have active participation from the user community and software delivery teams. To do this, this study investigates project statistics of items in Table 3.1. The project statics of interest are used as an evaluation criteria, extending (Zhang et al., 2019) and noted below:

Application change requests are requests from application users to the support community. Changes can be logged in GitHub using the inbuilt `Issues` tool. `Issues` can be ideas, feedback, bugs or feature requests (GitHub, 2022a). `Issues` counts for selected projects is gotten from the downloaded GitHub events data from GH Archive.

Developer activities are a series of activities carried out by the software development community that supports a GitHub hosted project and are indicators of the level of support and change rate for a project hosted on GitHub. Developer activities are made up of actions. Developer activities of interest in the evaluation of candidate cases are; the number of project contributors, forks, pull requests and code commits (push requests). Developer activities counts of `Forks`, `Push` and `Pull` events for selected projects is gotten from the downloaded GitHub events data from GH Archive. Contributors counts is gotten from the GitHub project page for each repository.

Community adoption is a measure that indicates the level of adoption and support of a project by developers and application users. This measure can be made up metrics such as the number of stars and watches a project has. A GitHub user can Star a project as a sign of interest (GitHub, 2022b) or appreciation (Padhma, 2021). Both watch and star counts are gotten from the GitHub project page for each repository.

Application Technology is a classification of the technology type of project being assessed. This helps us identify the technologies of an application such as the programming language it is written in, the function of the application and the operating system it runs on. The purpose of this criteria is to ensure selection processes are not bias towards a particular technology paradigm id est, platforms applications run on. For the purposes of this criteria, operating platforms that text editors run on are classified as OS or Browser. OS will indicate that the editor can be installed onto several operating systems such as MacOS, Linux or Windows id est, the editor is cross platform. Browsers indicate the editor can run on multiple browsers and again indicates that it is cross-platform. Operating platform information is gotten from the GitHub project page for each repository and the landing page for each project.

The evaluation criteria listed in Section 3.6.3 and corresponding search result statistics are listed Table 4.2.

3.6.4 Data Analysis

This section presents multivariate time series prediction as a research method used to predict software enhancements. The time series predictive model is the artefact produced by the DSR Design and Development process. Time series prediction is the process of creating a model to forecast future values using historical and current time series data as model inputs. Time series prediction can be short-term or long-term, with long-term predictions needing to cater for cumulative errors and a lack of information reducing the certainty of predicted outcomes (Sorjamaa, Hao, Reyhani, Ji & Lendasse, 2007).

Time series forecasting models can be univariate or multivariate, where univariate forecasting focuses on analysing and predicting an individual time series within a model, and multivariate focuses on multiple time series analysis and predictions within a model (Hahmann, 2019). Github events are time-stamped activities recorded against a software repository hosted on Github and, therefore, can be considered to be time and event-based datasets (Varuna & Mohan, 2019).

For the purposes of this research, Github events are used as variables that are investigated in answering the research questions in Section 2.7.1. Due to multiple variables in a time series data set being used to predict future outcomes, a multivariate method is used in this research. Multivariate analysis is defined as statistical techniques that analyse multiple measurements on objects being investigated (Hair, Black, Babin & Anderson, 2013). Multivariate analysis aims to create knowledge to allow quick decision making resulting in quicker reactions and the creation of high-quality products which deliver value to customers through the extraction of knowledge from information.

The process of time series analysis follows the steps to ensure the process of using data to forecasting values is structured, data and models are validated and forecasts tested for accuracy (Kumar Dubey, Kumar, García-Díaz, Kumar Sharma & Kanhaiya, 2021):

Explore data (exploratory analysis) Conduct an exploratory data analysis to gain insights into the structure and patterns of the data. This includes visualising the data, calculating summary statistics, and identifying trends and patterns. This is further detailed in Section 3.6.4.

Prepare data Ensure that data is appropriate for statistical analysis. Ensure there are no gaps, outliers or skewness in the data set. Also ensure that the statistical properties governing the behaviour of the data set to be analysed do not change over time id est, the data is stationary (Cryer & Kellet, 1991). This is further discussed in Section 3.6.4.

Model Selection and Parameter Tuning Choose a multivariate model and determine optimal model parameter values. For the purposes of this study, Seasonal Vector Autoregressive Integrated Moving Average (SVARIMA) is the statistical model used to predict the selection of software enhancements. Use statistical tests to determine model performance. Model performance in this context is the likelihood of a model to predict outcomes based on its data compared to the complexity of the model. This is further discussed in Section 3.6.4. The predictive model created is treated as an artefact created as part of the DSR Design and Development process illustrated in Figure 3.1 and as such is demonstrated and evaluated. The demonstration and evaluation process is further discussed in Section 3.7.

Model Validation Use the model to predict values and validate the accuracy of forecasts by using reserved validation samples. This is further discussed in Section 3.6.4.

Exploratory Data Analysis

This section describes the steps of exploratory analysis of data used in this study. This study carries out a graphical and statistical exploratory analysis of GitHub events. For the purposes of this study, GitHub events relevant to answering the research question are considered variables used in time series analysis.

The graphical analysis of variables in the time series data set is done by plotting the time series information in selected cases into Scatter Plots, Histograms and Box Plots. The reasons for using the aforementioned graphs are detailed below:

Time Series plots Time series plots can be used to identify trends in GitHub events data over time. This is done by plotting counts of all GitHub over a time period. Time series plots can help identify seasonality, gaps in data, outliers and trends in data (Aborass, Abu Hassan, Sahalash & Al-Rimmawi, 2022).

Histograms The frequency at which a single variable is observed within a data set is plotted as a histogram. This graphical representation gives indicators of the statistical characteristics of a variable such as the normal distribution of data. Normal distribution of data shows its' applicability to statistical analysis and so indicates the fitness of data sets for use in MVA techniques.

Scatter plots Relationships between variables are explored by using scatter plots comparing two variables. Pairwise comparisons of metric variables allow relationships and their strength to be evaluated graphically and also identify outliers in data.

Boxplots An understanding of the skewness and distribution of data is important in time series analysis. The distribution of data and identification of outliers within

time series data sets can be graphically represented in Boxplots and assist in identifying areas of further investigation (Aborass et al., 2022).

The statistical analysis of variables in the time series data set is done by calculating the following statistics for each variable relevant to this study (Fitrianti, Belwawin, Riyana & Amin, 2019):

Mean a calculation of the average value of a variable. It is used to check the dispersion of data and can be used in identifying trends or seasonality in time series data sets.

Variance a measure of data dispersion indicating the deviation of data within a data set from the mean. A variance is a non-negative number that increases in size as the deviation from the mean increases.

Median the middle value of data. Data points comparison to the median can indicate the likelihood of it being an outlier or influencing the skewness of data.

Standard Deviation measures the deviation of a data point from the mean. It can be used to identify patterns in data or help identify high levels of variability.

Minimum and Maximum values in a data set are used to identify outliers. This can help avoid biased results resulting from skewness of data or exceptional outliers influencing statistical modelling.

Skewness a numeric measure that describes the asymmetry of a dataset's distribution. Statistical analysis requires symmetric data distribution, meaning data points are evenly distributed to the left or right of a central point in a dataset. A positive skewness value indicates a dataset is right-skewed, meaning the right tail of the central point of a dataset contains more values than the left. Conversely, a negative skewness value indicates the data is left-skewed, meaning the left tail of the central point of a dataset contains more values than the right. A **zero** indicates perfect distribution where both sides of the central point of a dataset contain an

equal number of values. A skewness value less than **-1** indicates a dataset is highly skewed to the left side of the central point of a dataset. A skewness value greater than 1 indicates a dataset is highly skewed to the right of the central point of data. Skewness between **-1** and **-0.5** indicates moderate left-skewness and values between **1** and **0.5** indicates moderate right-skewness. Skewness values between **-0.5** and **0.5** indicates the dataset is relatively symmetrical. If data is moderately or heavily skewed, adjustments to the dataset are needed in order to use it for the purposes of statistical analysis.

Kurtosis a measure of skewness that measures the peakedness and flatness of a distribution relative to the normal distribution of a dataset. A Kurtosis value of less than **-1**, called a Platykurtic distribution, indicates the dataset analysed has flat peaks to the left and right of the central point of distribution when compared to the normal distribution. A kurtosis value of more than **1**, called a Leptokurtic distribution, indicates the dataset analysed has higher peaks to the left and right of the central point of distribution when compared to the normal distribution. A kurtosis value between **-1** and **1**, called a Mesokurtic distribution, indicates the dataset analysed is nearly normally distributed. If the kurtosis skewness of a dataset is not Mesokurtic, adjustments to the dataset are needed in order to use it for the purposes of statistical analysis.

Stationary Tests

This section presents the tests carried out against the time series data set used in this study to ensure it is stationary. Cryer and Kellet (1991) describes stationarity as no changes to the probabilistic laws governing a data sets behaviour over time. Stationarity can be tested by the Augmented Dickey-Fuller Test (ADF). ADF requires a null hypothesis to be formed as the data is non-stationary and returns a p-value and t-value.

The p-value calculates the statistical significance of the null hypothesis. If it is below 0.05 then the null hypothesis is rejected id est, there is little statistical significance to the null hypothesis. The t-value is used to calculate the p-value and measures how far away the sample mean is from the null hypothesis mean. The larger the t-value, the larger the evidence against the null hypothesis (Fitrianti et al., 2019).

If the data is found to be non-stationary, then differencing needs to be applied to the data set and the ADF test is re-run to confirm the stationarity of data.

Model Selection and Parameter Tuning

In this section, the Time Series analysis method is described. SVARIMA is presented as a multivariate time series statistical method used to predict software enhancements. SVARIMA is an extension of Vector Autoregressive Integrated Moving Average (VARIMA) supporting the modelling of seasonal components of time series data. Both SVARIMA and VARIMA are widely used statistical models used for multivariate time-series forecasting. Exploratory analysis of GitHub events data shows seasonality, therefore SVARIMA, which is a multivariate time-series model allowing modelling of season parameters, is used in this study (Malki et al., 2022).

VARIMA models are described by the equation *VARIMA (p,d,q)* where p, d and q are model parameters. The parameters *p, d, q* are the three main components of VARIMA describing Autoregression Term (AR), differencing and the Moving Average (MA) parameters of a model respectively. SVARIMA extends the VARIMA model by adding additional model parameters to account for seasonality.

An SVARIMA model can be illustrated by the equation SVARIMA *(p,d,q)* x *(P,D,Q)s* where *(p,d,q)* are non-seasonal parameters of a model discussed earlier. The *(P,Q,D)s* parameters are the seasonal parameters of a model extending VARIMA into SVARIMA to allow for seasonality. The parameter *s* is described as showing the period within a season (Kumar Dubey et al., 2021).

The selection of *(p,d,q)* and *(P,D,Q)s* parameters of a SVARIMA model need to be measured for goodness of fit to ensure the model parameters are tuned to select a model with a high likelihood of an accurate prediction and low complexity. This study uses the Akaike Information Criterion (AIC), the Bayesian Information Criterion (BIC) to measure goodness of fit as part of the models parameter tuning process (Mahdi, Provost, Salha & Nashwan, 2017). AIC and BIC calculate the goodness of fit for SVARIMA models by comparing the models likelyhood and the complexity of a model, where complexity is be determined by the number of parameters. BIC places a higher penalty on the complexity of a model than AIC.

This study compares multiple SVARIMA model parameters using AIC and BIC test results to select the most performant model. The optimal model will have lower AIC and BIC values as a lower score indicates that the tradeoff between the model complexity and fit to the data will result in the highest likelihood of accurate forecasts.

Model Validation

In this section, the process and statistical tests carried out against a forecasting model to evaluate model accuracy are discussed. For the purposes of this study the following process are followed:

Reserved data set Data from the time series data set that is set aside to allow comparison between forecasted values and actual events.

Statistical tests Statistical tests discussed later in this section are executed against models to evaluate model accuracy.

In this study, Root Mean Squared Error (RMSE), Mean Squared Error (MSE) and Mean Absolute Percentage Error (MAPE) are used to measure the accuracy of a forecast model (Q. Li, Zhao, Xue & Feng, 2021). RMSE is used to measure the size of the error between actual and predicted values. A zero value for RMSE indicates a perfect

fit; as a rule of thumb, lower values indicate a good fit. MSE measures the difference between actual data and forecasted values, where lower values indicate better model performance. MAPE measures the percentage of forecast error between actual data and forecasted values where lower values are considered indicators of better model performance (Kumar Dubey et al., 2021).

3.7 Demonstration and Evaluation

In this section, the process this study uses to validate an artefact created in Section 3.6. TODO: describe how a previously unused project will be put through the model and predictions will be compared to actuals etc.

3.8 Tools

SPSS Statistics a statistical analysis software tool created by IBM. This tool is used to carry out the examination of data and statistical tests. `https://www.ibm.com/spss`

Visual Studio Code a free, open-source source code editor developed by Microsoft. Visual Studio Code is used in this study as an efficient text editor to examine large CSV files containing GitHub events datasets. `https://code.visualstudio.com/`

Visual Studio a licensed source code editor developed by Microsoft that is used in this study to write code in. `https://visualstudio.microsoft.com/downloads/`

csharp an object oriented programming language developed by Microsoft that is now open-sourced. This is used in this study to create the GitHub ETL artefact described in Secton 4.5. `https://dotnet.microsoft.com/en-us/`

languages/csharp

Debian an open-sourced Linux based operating system used in this study to execute downloading of data from GH Archive. https://www.debian.org/

Bash scripts shell scripts run on a Linux environment used to download data from GH Archive.

PostgreSQL an open-sourced database used to store and analyse data as described in Chapter 4. https://www.postgresql.org/

pgAdmin a SQL query tool developed for PostgreSQL database used to execute SQL commands from. https://www.pgadmin.org/

Power BI a Data visualisation tool used to summarise data as it is imported to see if import logic is correct and if linking of data for selected projects was occurring correctly. https://powerbi.microsoft.com/

Chapter 4

Analysis

4.1 Introduction

In this chapter, the analysis conducted to answer the research questions of this study are presented. In Chapter 2, the need for taking a data-driven approach to identifying the software enhancements using an evidence-based approach is identified and DSR is selected as a research method used to answer this study's research questions. The findings from this study produce guidelines and models that software delivery teams can use in their decision-making processes which can be categorised as validated artefacts produced by the DSR process.

To answer the research questions and achieve the research goals specified in Section 3.5, data is collected from GH Archive and analysed. GH Archive is a repository of GitHub events data which captures software developer and user interactions against open-source projects used for research purposes. The data allows researchers to perform time series analysis and gain insights into the evolution of software projects hosted on GitHub over time.

By applying statistical analysis against GitHub events, an understanding of the SDLC can be distilled and predictive models can be developed. The aforementioned

outputs can be generalised and published as artefacts which can be used by software delivery teams in making informed decisions regarding software enhancements.

In this chapter, the process of collecting data for analysis, which involves downloading data, validating data, understanding data and creating a data model that is used for analysis is discussed in Section 4.4. The exploration of data to find trends and relationships and ensure it is suitable for statistical analysis is discussed in Section 4.7. Section 4.8 describes the creation and validation of an SVARIMA predictive model that can be used by software development teams in making evidence-based decisions regarding software enhancements.

4.2 Search GitHub for Repositories and Shortlist Candidates

The search results based on GitHub search criteria specified in Section 3.6.3 are listed in Table 4.1.

Table 4.1: Project search results

Repository Name	**Stars**
microsoft/vscode	134k
atom/atom	56.2k
marktext/marktext	33.2k
quilljs/quill	33k
microsoft/monaco-editor	30k

The results from the search criteria are further examined by evaluating project statistics discussed in Section 4.3.

4.3 Evaluate Candidates

In this section, the evaluation of shortlisted GitHub projects noted in Section 4.2 are evaluated as cases to be used in this paper. Evaluation is carried out using the project statistics discussed in Section 3.6.3. The evaluation criteria listed in Section 3.6.3 and corresponding search result statistics are listed Table 4.2.

Table 4.2: Project selection criteria

Project statistic	**vscode**	**atom**	**marktext**	**quill**	**monaco**
App change requests					
Num of `Issues`	144,432	8,358	3,701	1,914	2,966
Developer activities					
Num of `Forks`	19,047	15,145	3,210	2,252	1,978
Num of `Push`	53,188	4,128	2,120	386	775
Num of `Pull` Requests	18,449	3121	1,917	361	321
Num of Contributors	1,849	495	140	134	232
Community adoption					
`Stars`	134k	56.2K	33.2K	33K	30K
`Watches`	3,252	2,632	395	472	517
App Technology					
Operating platform	OS	OS	Browser	Browser	Browser

The project statistics captured in Table 4.2 show that the cases selected are available to multiple technology platforms as they are either cross-platform compatible from an operating systems perspective or run on browsers which are also cross-platform compatible. The projects have user community of feedback in the form of `Issues` logged and developer activities relating to code changes are occurring. The statistics also show that users are indicating interest in the project due to the number of `Stars`

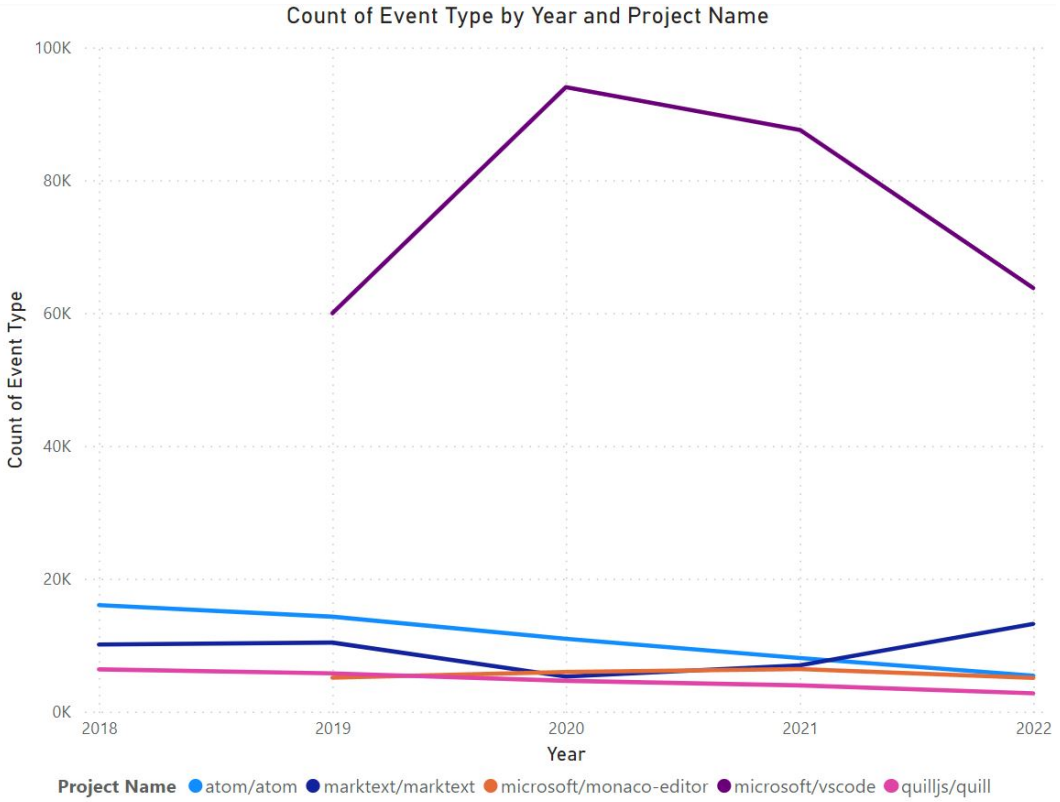

Figure 4.1: GitHub activity counts by year

being given and `Watches` being assigned over time. Figure 4.1 shows a time series plot of Github events listed in Table 4.2 (`Issues`, `Forks`, `Push`, `Pull` and `Watch` events). The time series plot shows continual activities against the selected cases by both users and contributors of the GitHub projects.

In summary, Table 4.2 shows that the cases selected for analysis are active with the developer and user community interactions and are popular due to the `Stars` and `Watches`. The time series plot illustrated in Figure 4.1 shows that the activities against the projects are ongoing. As such the cases selected for analysis show signs of continual activities.

4.4 Data Collection

In this section, the process of collecting GitHub events from GH Archive, the primary data source, is discussed. GH Archive is a research tool that collects and archives publicly available events from GitHub on a daily basis (Jarczyk, Gruszka, Jaroszewicz, Bukowski & Wierzbicki, 2014). The methods available from GH Archive for downloading data are considerably easier than GitHub as there is no need to call multiple APIs, resulting in simpler download scripts, meaning less development time is required. The speed of data acquisition from GH Archive is faster than GitHub as there are less data throughput limitations on GH Archive download processes which results in faster download times for large data sets. Lastly, the data models returned from GH Archive files are denormalised, which results in simpler queries needed for the analysis of data.

GH Archive organises publicly available datasets as JSON files that contain GitHub events data and relevant information relating to the project and GitHub user these events are related to. The files are organised by date and hour and can be downloaded using a publicly available API (ProgrammableWeb, n.d.). The API url structure pattern used is `https://data.gharchive.org/{year}-{01..12}-{01..31}-{0..23}.json.gz` where **{year}** represents the year of the data set, **{01..12}** represents the month range for the data set, **{01..31}** represents the day range of the dataset and **{0..23}** represents the time range of the data set. For this study, data sets from the 1 January 2018 through to the 18 September 2022 are downloaded, which is the date the analysis of data started.

Using the GH Archive public API allowed data downloads to be automated. The download of data used the **wget** commands in a bash script that is executed in a Debian CLI. Listing 4.4 shows a code snippet from the download script run to pull all GitHub events from GH Archive for 2021.

```
1 #!/bin/bash
```

```
2 cd /mnt/e/gharchive
3 mkdir 2021
4 cd /mnt/e/gharchive/2021
5 wget --continue https://data.gharchive.org
     /2021-{01..12}-{01..31}-{0..23}.json.gz
```

Listing 4.1: 2021 dataset download script example

The complete download script for the snippet shown in Listing 4.4 is available on GitHub at the following URL `https://bit.ly/42rOq3W`.

Once downloaded, an inspection of the JSON files is carried out to determine the data model of the files. During this process, it is found that the datasets from GH Archive aggregate disparate GitHub events and their individual data structures into one easy-to-analyse structure. This means that each line in the GH Archive data set represents a JSON object containing information about a GitHub event that occurred against a repository (Du et al., 2020). A description of GitHub events contained in the downloaded GH Archive data files is presented below:

CommitCommentEvent A comment added to a code commit which is a `PushEvent` in GitHub.

CreateEvent Creation of a code repository or a branch of code within a repository. A `CreateEvent` makes a copy of code within a repository that has changes applied against it. This is different to a `ForkEvent` where a branch of code is created but external to the original repository.

DeleteEvent Deletion of a branch of code, tag or another resource.

ForkEvent Creation of a copy of a repository. Typically used to clone a repository to another GitHub user account, similar to creating a branch of code external to the original repository, to help isolate changes made by developers. Changes are reviewed and merged into the main branch of code after a code review

has occurred and the changes are approved. The request for code review is a `PullRequestEvent`.

GollumEvent An update to a repositories wiki which is the documentation of a repository.

IssueCommentEvent A comment added to an issue.

IssuesEvent An event requesting a change to code. These can be bug reports, feature requests or labels identifying milestones of a project.

MemberEvent An event related to the change in membership to a repository. These can signify events such as the removal or addition of collaborators or changes to their roles.

PullRequestEvent An event signifying that a code change contained in a `Fork` is presented for review and acceptance into the main code base of a repository. A `Pull Request` typically triggers a `PullRequestReviewEvent` that is carried out by a developer. Once completed, a `Pull Request` is closed and the associated `Fork`, if one exists, is deleted.

PullRequestReviewCommentEvent A comment added to a `PullRequestEvent` noting any changes made to the `Pull Request`.

PullRequestReviewEvent An event that is created when a `Pull Request` is reviewed by a project contributor such as a developer. This process typically involves a developer reviewing code changes and accepting or rejecting them. Once a `PullRequestReviewEvent` is completed, a `PullRequestEvent` is closed, if the changes are accepted, and the `Fork` which the `PullRequestEvent` relates to is deleted after the code is merged into the main codebase.

PushEvent Code changes pushed to a branch of code contained in the `Fork` or if no fork exists then changes are pushed to the main codebase. Once code changes are pushed to a `Fork` and the developer is ready to merge the changes to the main codebase, a `PullRequestEvent` is triggered.

ReleaseEvent A `ReleaseEvent` is created by projects when a new version of code is ready for release. This isn't always used but some projects use this functionality to manage multiple versions of code releases available to their users.

WatchEvent An event that occurs when GitHub users want to track changes to a repository hosted on GitHub. `WatchEvents` can be thought of as subscriptions to repositories where notifications of changes to code can be sent to watchers.

An investigation of data files is carried out to determine the GitHub events data model. The JSON Viewer tool is available at `https://bit.ly/43PoyzX` and is used in this process. JSON payloads from the dataset for different event types described in Section 4.5 are copied into the tool, generating a Tree view. The tree view shows a data model with a parent object and nested child objects that exist within each JSON record. An sample of the JSON Tree view of a `PushEvent` is shown in Figure 4.2.

The parent object contains general information relevant to all GitHub events, such as an `event id`, event type description, an actor object representing the user that triggered the event, an event created date time stamp and the repository the event occurred against. These are items surrounded with red boxes in Figure 4.2. Findings also indicate that each specific GitHub event described in Section 4.5 has a unique nested data structure within the event containing information specific to each event type. After investigating the attributes of each GitHub event in the data sets, a data transformation script that created the high-level data model described in Figure 4.3 is created.

`https://bit.ly/3J5w7L5` contains the data transformation script created to simplify the process of importing data into a database and make subsequent querying of imported data for analysis easy. The data transformation process is noted as Step 2 in Figure 3.4 and is part of the import process discussed in Section 3.6.2. Once transformed, data is prepared and imported into a PostgreSQL database for further

Figure 4.2: JSON Tree View of a GitHub PushEvent

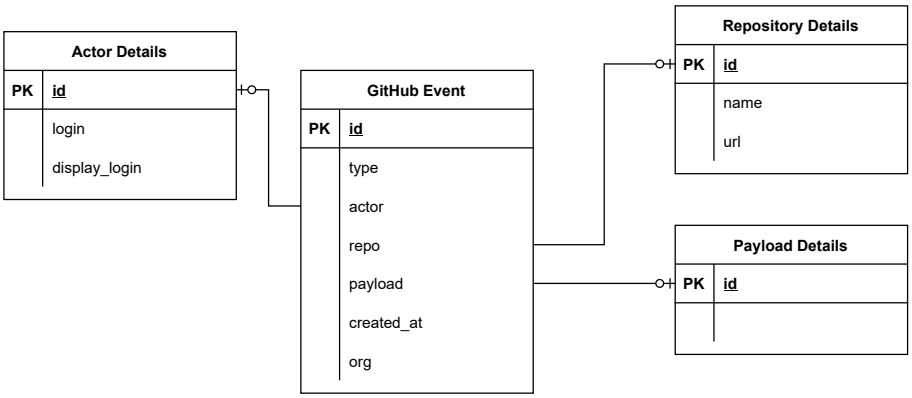

Figure 4.3: GitHub Events high-level data model

analysis. This process, noted as Step 3 in Figure 3.4 is further detailed in Section 4.5.

4.4.1 GitHub Analysis Data Model

In this section, a summary of the data model used for the analysis detailed further in this chapter is discussed. The data model described is a summary of the detailed data model used in this study, highlighting key fields relevant to the analysis carried out in this chapter. For the purposes of future reference, this data model is referred to as the GitHub analysis data model.

Figure 3.4 illustrates the transformation of source file data models into an intermediate model, which is processed by the data import application. The data import application further flattens the transformed data to simplify structures for subsequent processing, storage and querying. The GitHub analysis data model from which database tables are created from is illustrated in Figure 4.4.

The fields of the GitHub analysis data model fields shown in Figure 4.4 are noted Section 4.4.1 below:

Id A unique identifier assigned to the row of data.

file_name The name of the source file the row of data originated from.

row_number The row number of the source file the data originated from.

event_id The unique identifier of a GitHub event assigned to events occurring against a repository hosted in GitHub. Specified as the object id in Figure 4.2. A GitHub `event id` can have multiple event types associated to it. A GitHub `event id` allows events to be referenced to each other for timeline tracking purposes, id est, it tries to correlate multiple GitHub event types to specific events.

event_type The classification of an event that occurs against a repository hosted on GitHub. Examples of event types are detailed in Section 4.4.

repo_name The name of a repository hosted on GitHub which events occur against.

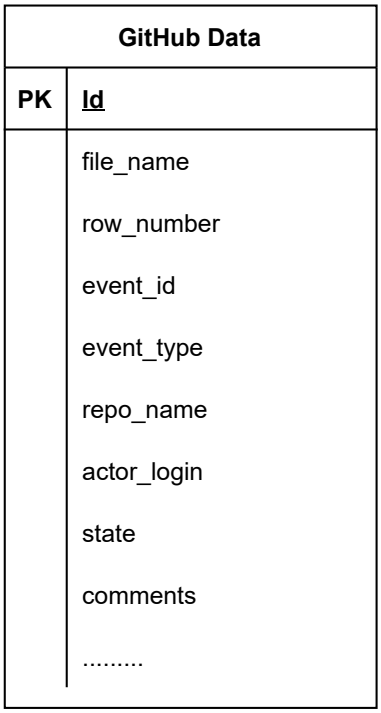

Figure 4.4: GitHub analysis data model

actor_login The login of the user who created an event against a repository hosted on GitHub.

state The state of an event against a repository hosted on GitHub. The state can be open or closed id est, a `Pull Request` is open awaiting approval and closed when reviewed and approved.

comments A comment associated with an event that occurs against a GitHub repository. This can contain useful information such as references to GitHub and other events related to an event a comment is against.

4.5 Data Preparation

In this section, the process of validating, cleaning and adjusting data for the purposes of importation for analysis is discussed. The importation process, noted as Step 3

in Figure 3.4 is an iterative process illustrated in Figure 4.5. The details of the data preparation and import process steps are detailed below:

Load transformed files to memory This step in the import process involves reading files from the transformed data staging folder into a list of data objects in memory. The subsequent steps of cleaning and writing data into a database use these lists of objects. The details of this step in the process are discussed further in Section 4.5.1.

Clean data Execute data cleaning as required. Examples of these activities are removing invalid characters, converting or reformatting data types or building missing relationships between data objects that are imported. The details of this step in the process are detailed further in Section 4.5.2.

Load data from memory to database Bulk import data into a database. The details of this step in the process are defined further in Section 4.5.3.

Validate imports Use SQL scripts and data visualisations to evaluate imported data for correctness. The details of this step in the process are detailed further in Section 4.5.4.

Make corrections and re-run imports Results of the validation steps carried out at each stage of the data import process may require changes to the transformation, cleaning or loading logic. Therefore, as changes are made, the import process may need to be re-run. This results in the data preparation and import process needing to be iterative, where each iteration is an improvement on the last version.

4.5.1 Load transformed files to memory

In this section, the process of loading transformed GitHub event data into memory for further cleansing and processing is discussed. Due to the large number of files to process and the large size of individual files, a batch import process is used. The batch

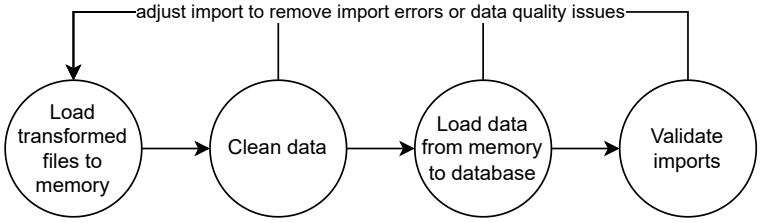

Figure 4.5: GitHub data preparation and import process

import process iterates through each data file in the transformed data staging folder, and reads it to a data object in memory, grouping records by day of the year. This ensures that in the event of a processing error occurring, the import process can continue based on the last day successfully imported.

4.5.2 Clean data

In this section, the data-cleaning steps used in cleansing information read from import files are detailed. During the import process, invalid characters in GitHub data are found, resulting in failure to read data from source files and an inability of tools to interpret data or persist data in databases. Control characters, which are characters that cannot be displayed as text but can be read as code instructions, are found in the data. Unicode characters also exist that are incompatible with the technologies used in the processing, storage or reporting of GitHub data used in this research. Special characters are also found that are identified as scientific symbols or emojis. The import process is adjusted to strip control and Unicode characters from data as they are read from source files. Special characters are stripped, or fields containing them are ignored if they are not relevant to the analysis requirements of this project.

4.5.3 Load data from memory to database

In this section, the process of writing cleansed data from memory to PostgreSQL in order to run subsequent data analysis is highlighted. Due to the large volume of data

files and rows of data being processed, a batch database import process is used. An iterative process is used to identify the database batch import sizes, which involves testing the duration of imports for different batch import sizes. Import batch sizes of 500, 100, 500, 10,000, 50,000, 100,000 and 200,000 rows of data are tested, and the database write times for batches are recorded in logs.

Experimentation results indicate that the optimal batch import size is 100,000 rows of data. Log data analysis shows that each file's average import time slowly increases in duration. Database performance degradation due to the growth in database size where file import times increased as database size increased. A strategy of partitioning database tables by date and time is tested with minimal gains and abandoned as it added to the complexity of the data import process with import times savings being achieved.

4.5.4 Validate imports

In this section, the process of validating imported data for correctness is discussed. The data validation process contains three steps to assist in identifying data issues. First, summary counts of imported data are generated to identify the number of rows of data in source files and the number of rows of data written to the database. This helps ensure that there are no data write errors resulting from records from source files being dropped.

As discrepancies between source file record counts and database record counts are identified, records ignored during the import process are identified, and troubleshooting steps are carried out to find the causes. Through troubleshooting, the sources of data import errors resulting in records not being imported from the source files are identified and corrected. Most corrections are needed due to issues detailed in Section 4.5.2, and so the data cleaning logic is updated and importing re-run across several iterations. Import process improvements are stopped once import accuracy reached the point of 98

% of source file records being correctly imported into the database.

Due to the nature of the research question, this study needs to ensure that relationships between GitHub events are correctly identified and stored in the database. Examples of relationships between GitHub events are bugs in repositories being logged that may receive community comments or feedback and then have subsequent software development activities against them, resulting in code commits, code reviews and releases. Using the terminology noted in Section 4.4, the events in the previous sentence may translate to an `Issue` event occurring, leading to `Issue` Comment events, `Fork` events, `Push` events and `Pull Request` events. As such, the relationship between GitHub events being retained during the import process is tested.

Power BI data visualisations are used as a tool to graphically render GitHub events and their relationships, and inspect and confirm that relationships are retained when data is imported from source files into the database. The visualisation created is a network diagram of `event ids` as source nodes of the network and event types as the target node. The visualisation created illustrates relations, as connections, between events and corresponding types of events demonstrating relationships between a GitHub events as a parent event that occurs with the types of events associated to it. If events and event types have relationships in the network diagram, then the import process has captured relationships correctly. If not, then data is to be further investigated to find why orphaned events exist.

The first successful GitHub event import shows the majority of events in the database are orphaned due to a lack of connections between `event ids` as shown in Figure 4.6. The red box encloses an example of an `event id` node, `event id` nodes in the network diagram are represented by grey dots. Event types are represented as the coloured nodes in the network diagram, an example of which is enclosed by the green box. Due to the interrelated nature of Github events, we expect to see relationships between `event ids`, however Figure 4.6 shows that `event ids` are independent.

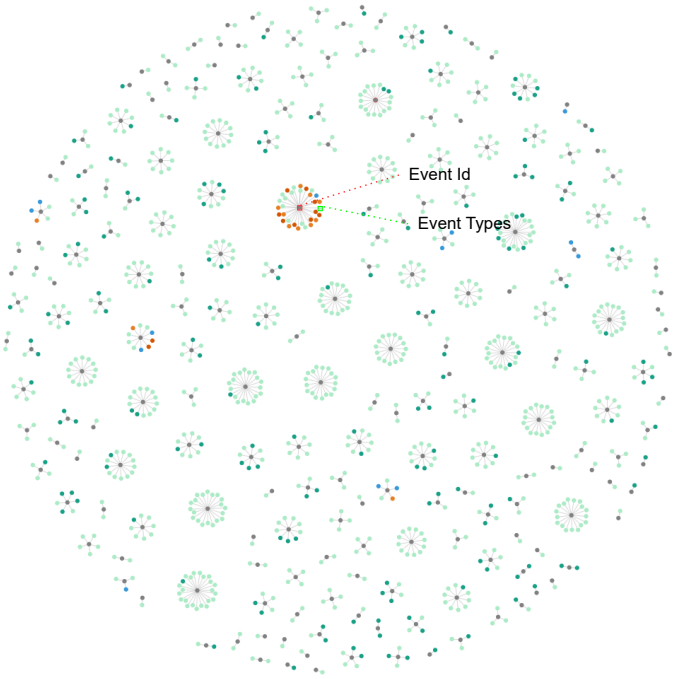

Figure 4.6: Initial Event Id and Event Type Network Diagram

Spot checks of records in the source files shows that the lack of relationships between events occurred due links between `event ids` being contained in the comment attribute of the source JSON records. To resolve this issue, the import process is adjusted to scrape the comment fields for each GitHub event record to identify URLs referring to GitHub APIs related to the event types described in Section 4.4. The import process is adjusted over several iterations with logic changes and the final network diagram generated shows relationships between `event ids` formed with no orphan records, as shown in Figure 4.7.

4.6 GitHub Events Sequence

In this section the core activities of software development and their mappings to GitHub events are detailed. The summary of GitHub events, as noted in Section 4.4, can be used to indicate an expected order in which GitHub events occur. The sequence of Github

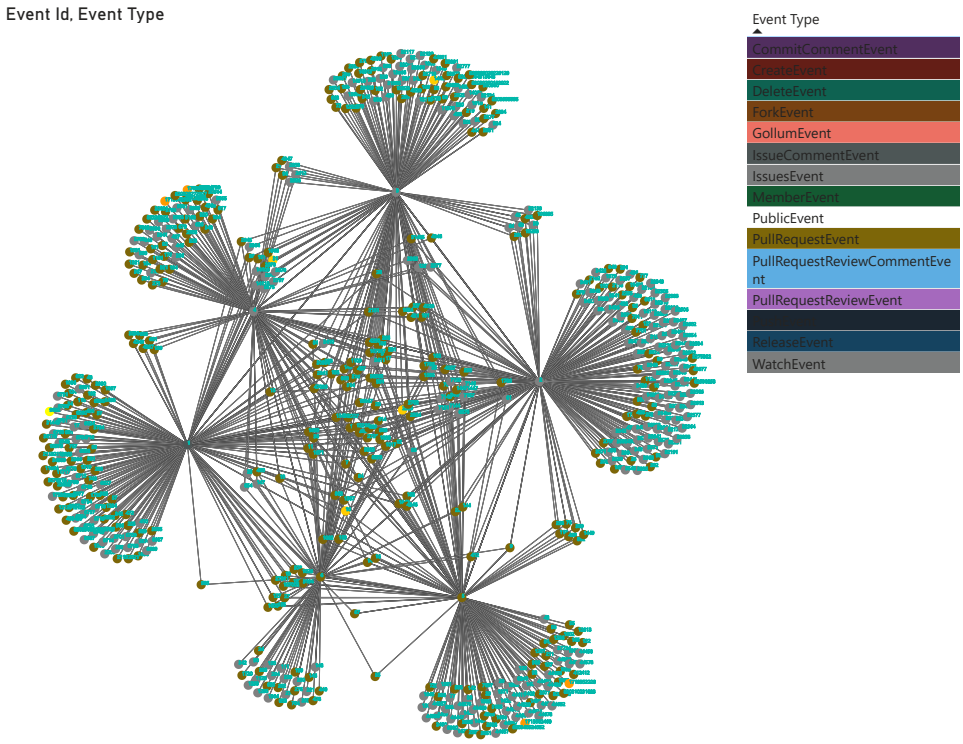

Figure 4.7: Second Event Id and Event Type Network Diagram

events in relation to the software development life cycle is used in this chapter to help understand the results of statistical analysis of the sample population datasets.

Software changes start with a request for change to a code base that can be a feature enhancement or a bug report. GitHub collects bug report or feature request as `Issues` which are accepted by the software delivery teams and result in code changes. Code changes can occur within the master code base (represented by `Push` events), an isolated clone of the master repository (represented by a `Fork` event), or a new branch of code within the same repository (represented by a Create event). Software delivery standards and practices determine which of the previously mentioned methods are used in the process of making code changes. Common practices involve making a branch of code within the repository so lineage of code changes are visible and easy to manage, and so it is expected that an `Issue` event would be followed by a Create event. With open source projects, it is expected that code may be changed in a clone of the master repository due to inconsistent practices introduced by a distributed team working at different skill levels or adhering to differing standards and practices. As such, it is expected that `Issue` events may also be followed by Create or `Fork` GitHub events.

With the creation of a copy of source code through a clone (`Fork`) or branch (Create), developers can start creating changes and committing them for source control purposes. These activities are represented by `Push` events where changes are pushed from developers into a change sets. As such it is expected that Create or `Fork` event are followed by `Push` GitHub events.

Change sets, represented by `Push` events, may directly update the master code base however common practices and standards specify that a series of changes from `Push` events need to be combined into a `Pull Request`. `Pull` Requests are a set of changes that are not currently part of the main code based and need peer review and approval to be accepted and subsequently merged into the master code base. `Pull` Requests are represented as `Pull Request` events in GitHub and are created by

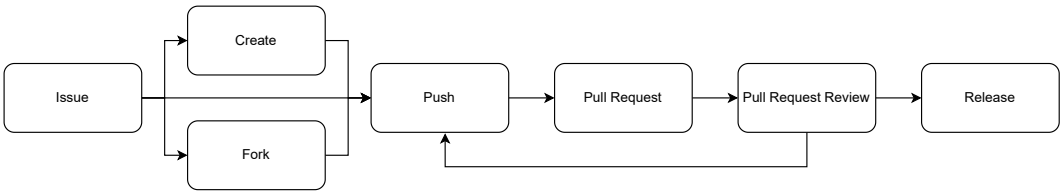

Figure 4.8: GitHub events sequence

software developers who have made a series of code changes that are pushed to GitHub. As such it is expected that `Push` events ware followed by `Pull Request` Events.

The process of peer reviewing `Pull` Requests result in `Pull Request` Review events being created with associated comments and an acceptance or rejection of code changes. Rejection of `Pull Request` events can result in additional `Push` events as issues in the code changes are identified remediated. It is expected that successful `Pull Request` events are be followed by `Pull Request` Review or `Push` GitHub events.

Once a `Pull Request` is approved by a `Pull Request Review` event, changes to code are merged into the master code base so new features or remediation of bugs can be released to users. The release of code to users is represented by a `Release` GitHub event and so it is expected that approved `Pull Request Reviews` result in `Release` GitHub events.

The process described above, summarises the key software development activities of the delivery lifecycle which is illustrated in Figure 4.8. The statistical relationships between these relationships are further explored in this chapter.

4.7 Descriptive Statistical Analysis

In this section, the process of executing the exploratory analysis detailed in Section 3.6.4 and their findings are presented. The data used for analysis is acquired using the methods discussed in Section 4.4 and prepared for analysis using processes specified in

Section 4.5. The data model used for analysis is illustrated in Figure 4.4.

Statistical analysis is performed against a subset of GitHub repositories in the collected dataset. Due to the large population size of the collected dataset, a subset of repositories are used to reduce resource requirements and duration of analysis. This subset of repositories are cases identified in Section 4.1 and evaluated for use in Section 3.6.3.

The exploratory statistical analysis carried out in this study start with examining the distribution of GitHub events using methods specified in Section 4.7.1. Then examination of the temporal characteristics of GitHub events is executed in Section 4.7.2. The relationships between GitHub are then explored using methods detailed in Section 4.7.3.

4.7.1 Distribution and Skewness of GitHub Events

In this section, the skewness and distribution of GitHub events are examined to explore the characteristics of data. This is done to evaluate the suitability of case datasets for statistical analysis, as distribution and skewness play a critical role in statistical analysis. The validity and reliability of data are evaluated through Histograms, and summary statistics discussed further in this section.

Histograms provide graphical representations of the frequency and distribution of data. An examination of the frequency of GitHub events can determine the symmetrical distribution of data along with deviations. Deviations in data result in skewness or outliers visualised in histograms.

Summary statistics are used to complement histograms in qualifying the distribution and skewness of GitHub events. Summary statistics used in the evaluation of GitHub events are mean, median, variance, standard deviation, minimum, maximum and skewness. These are further defined in Section 3.6.4.

Figure 4.9 illustrates the frequency of GitHub events in the dataset for the cases

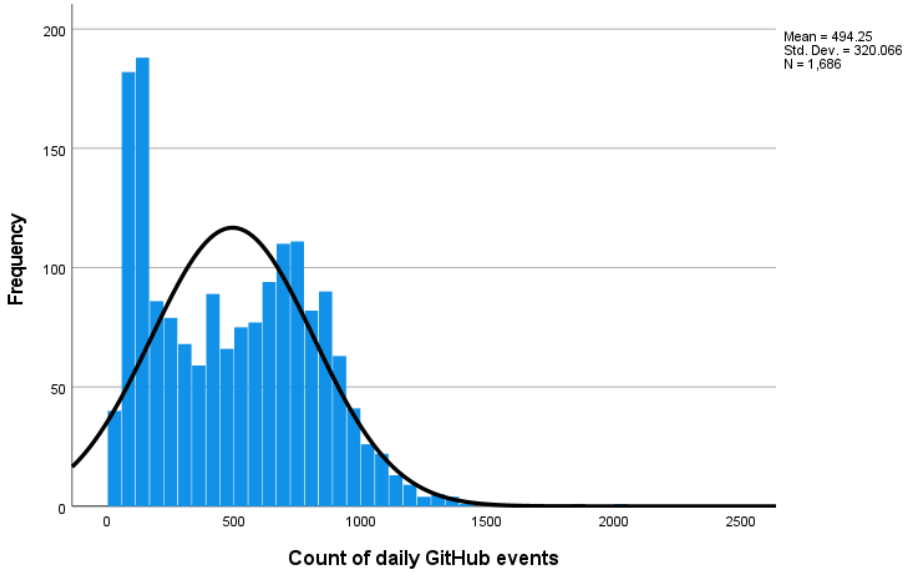

Figure 4.9: Frequency of daily GitHub events

selected for analysis in Table 4.1. To generate the histogram, all GitHub events are grouped to calculate a count of all events that occurred daily for the cases selected in Section 3.6.3 using SPSS. The visualisation shows that the majority of data is within the normal distribution curve. There seems to be a slight positive skew in the data indicating a slight right skew as denoted by the normal distribution curve having a longer right tail when compared to the left. Summary statistics are then used to further investigate skewness.

The summary statistics noted in Table 4.3 support the findings of the visual examination of the histogram illustrated in Figure 4.9. This study observes that the minimum number of GitHub events in a day is 56027, meaning that the number of GitHub events per month below this value would be unlikely. The maximum number of GitHub events observed is 85805, meaning that the number of monthly GitHub events are unlikely to be more than this value.

The mean, a measure of the average number of GitHub events in a month, is 494.25, and the median 490.50. The mean and median values are indicative of the central point

in thee data, meaning that the typical number of GitHub events occurring every month is likely to be close to 494.25 and 490.50. Median is less sensitive to variations in data introduced by extreme values, and so the proximity between the mean and median values are indicative of the dataset not being heavily skewed.

The skewness of data is observed to be 0.347. The result indicates a slight right-skewness in the dataset; however, this is within -0.5 and 0.5, meaning the dataset analysed is relatively symmetrical and, as such, is normally distributed. The kurtosis value of data is observed to be -0.575. This indicates a mesokurtic distribution as it falls within -1 and 1, meaning the data analysed is nearly normally distributed.

The percentiles of data described in Table 4.3 provide an indicator of data normality. The 50th percentile is observed to be 490.50 and is the same as the median of 490.50. This indicates that data is evenly distributed around the central point of the dataset where 50 %, of data is above the central point and 50 % below it. The distance between the 25th percentile and the mean is 322.5, and the distance between the 75th percentile and the mean is 249.75. The distance between the mean and percentiles is similar but indicates the dataset has a longer tail on the right side of the distribution and data points are more concentrated on the left side of the distribution.

Standard deviation describes the dispersion of data around the mean. The standard deviation of the dataset is observed to be 320.066. When compared to the mean, 494.25, the standard deviation shows a high dispersion of data from the mean. As a result, the data shows signs of variability that further statistical analysis investigates.

Summary statistics captured in Table 4.3 indicate the dataset analysed is symmetrical. The skewness and kurtosis values support the graphical observation that the data is slightly right-skewed but still normally distributed, meaning the data appears to be relatively symmetrical. The distance between the 25th and 75th percentiles and the mean shows data is concentrated on the left side of the normal curve. The small difference between the mean and median values indicates that the dataset is not influenced by

extreme values in the dataset. The histogram illustrated in Figure 4.9 does not show a high level of skewness in the data and a relatively symmetrical distribution with a concentration on the left and a long right tail with data mostly within the normal curve. As such, the dataset being analysed appears to be normally distributed.

Table 4.3: Distribution summary statistics

Summary Statistic	**Value**
Minimum	1
Maximum	2039
Mean	494.25
Median	490.50
Variance	102442.482
Std. Deviation	320.066
Skewness	0.347
Kurtosis	-0.575
25th Percentile	171.75
50th Percentile	490.50
75th Percentile	744

4.7.2 Temporal Characteristics of GitHub Events

In this section, the temporal characteristics of GitHub events are examined. The temporal characteristics of interest in the context of this study, as noted in Section 3.6.4, are gaps in data, seasonality, trends and outliers. The rationale for investigating the previously mentioned characteristics and findings are detailed further in this section below.

To investigate the temporal characteristics of the dataset, time series visualisations

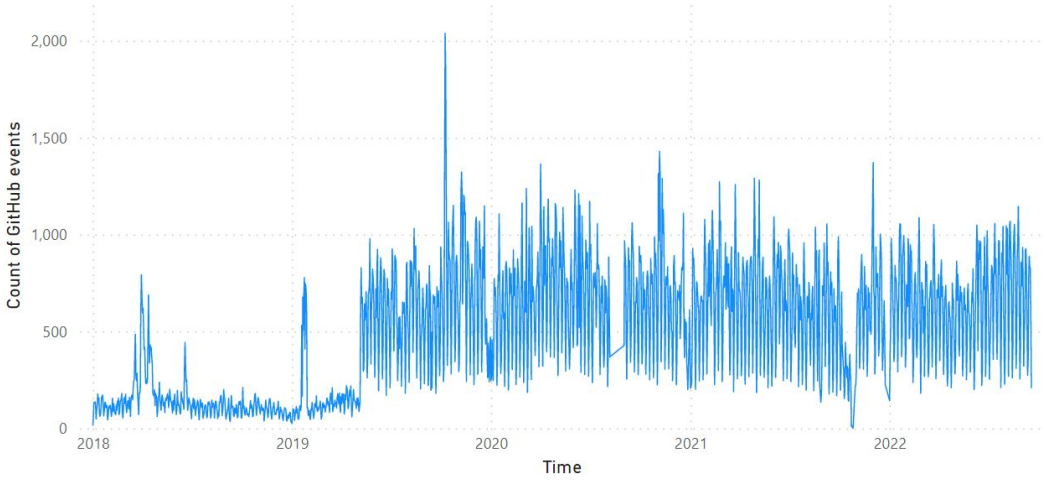

Figure 4.10: Counts of all daily GitHub events

are used. Figure 4.10 illustrates a time series plot of the selected cases' daily GitHub event counts. Figure 4.10 shows a high level of variability in counts between days, supporting the standard deviation statistic noted in Table 4.3. A spike in the number of daily events on the 9th of October 2019 is shown, where the daily count of GitHub events reached 2039, which was above the 75th percentile of counts in the summary statistics.

Figure 4.10 also shows that the number of Github events increased noticeably after the first quarter of 2019. To investigate this increase in GitHub event counts, daily GitHub event counts are split by the cases inside of the dataset analysed, which are represented as GitHub repositories as illustrated by Figure 4.11.

Figure 4.11 shows that the increase in daily GitHub events previously mentioned is due to Visual Studio Code, noted as Microsoft/vscode, activities being added to the dataset on the 8th of May 2019. The visualisation also shows that when compared to the other cases in the dataset, Visual Studio Code has a higher number of daily activities on GitHub, resulting in larger daily GitHub events observed in Figure 4.11 as previously noted. Visual Studio Code also appears to have a larger variation in daily GitHub events occurring when compared to other cases. Although Visual Studio code is added on the

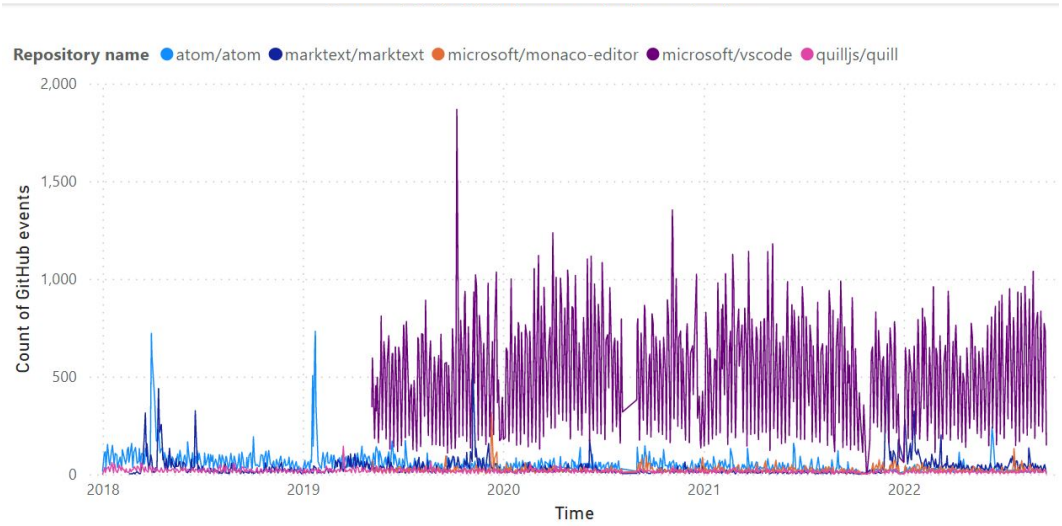

Figure 4.11: Counts of all daily GitHub events by Repositories

8th of May 2019, the data set contains three years of data points meeting the study case selection criteria.

Figure 4.11 also shows that there is a smooth increase in GitHub events for all cases from the 6 August 2020 to the 1 September 2020. The smooth increase could be due to missing data on all events or a subset of events for the previously identified time period. To investigate further, a time series visualisation, illustrated in Figure 4.12 is created. Figure 4.12 illustrates GitHub event counts over time categorised by GitHub event types with the purpose of identifying the type of events data points missing. Figure 4.12 shows that all event types for all cases are missing events from the 6th of August 2020 to the 1st of September 2020.

Figure 4.12 illustrates a decrease in Github events for all cases in October 2021. A similar approach to investigate the data, as described in the previous chapter, is taken to investigate the data trend. Figure 4.14 illustrates all GitHub events for the sample population filtered to show data from 1 October 2021 to 1 December 2021. Figure 4.14 shows a drop in the number of GitHub events from the 23rd of October 2021 to 25f October 2021. Investigation finds that data is still being captured for that time period,

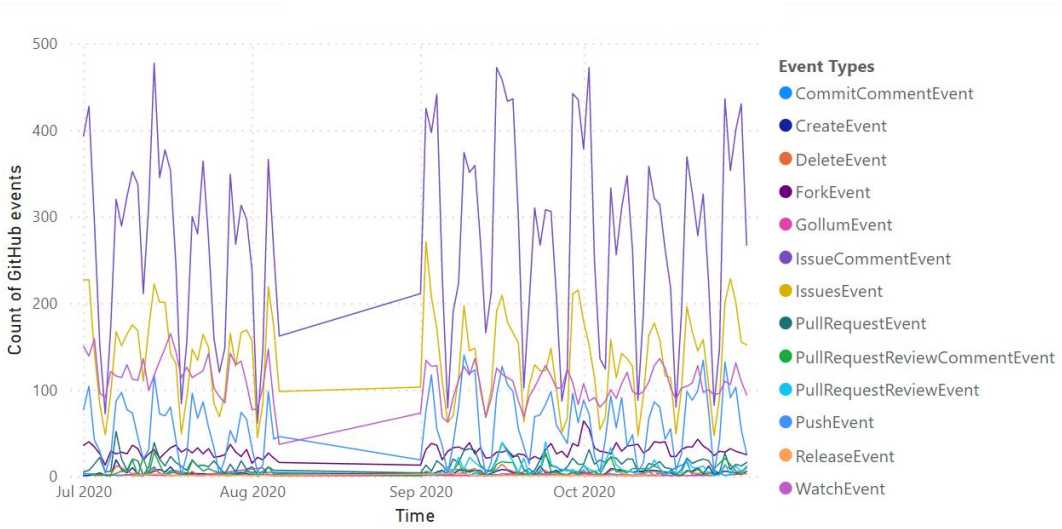

Figure 4.12: Counts of all daily GitHub events by Event Types September 2020

however, the volume of events generated for the cases in the sample population is lower than the days immediately before or after this window of time. A summary of the GitHub events for all cases from 1 October 2021 to 25 October 2021 is noted in Section 4.7.2. This indicated a trend that may exist due to the seasonality of the dataset, which is further investigated in this section.

23rd October 2021 1 commit comment, 3 issue comment, 1 issue, 1 push and 2 watch GitHub events are present in the sample population dataset.

24th October 2021 4 commit comment, 3 issue and 4 watch GitHub events are present in the sample population dataset.

25th October 2021 3 issue comment and 2 watch GitHub events are present in the sample population dataset.

Missing data from August to September 2020 could not be explained through investigation of the source data files. Log analysis of the import process does not show any recorded import issues. The same investigation efforts for missing records from 23 October 2021 to 25 October 2021 does not present explanations. Due to the inability to explain the missing data and the small volume of missing data and the summary stastics

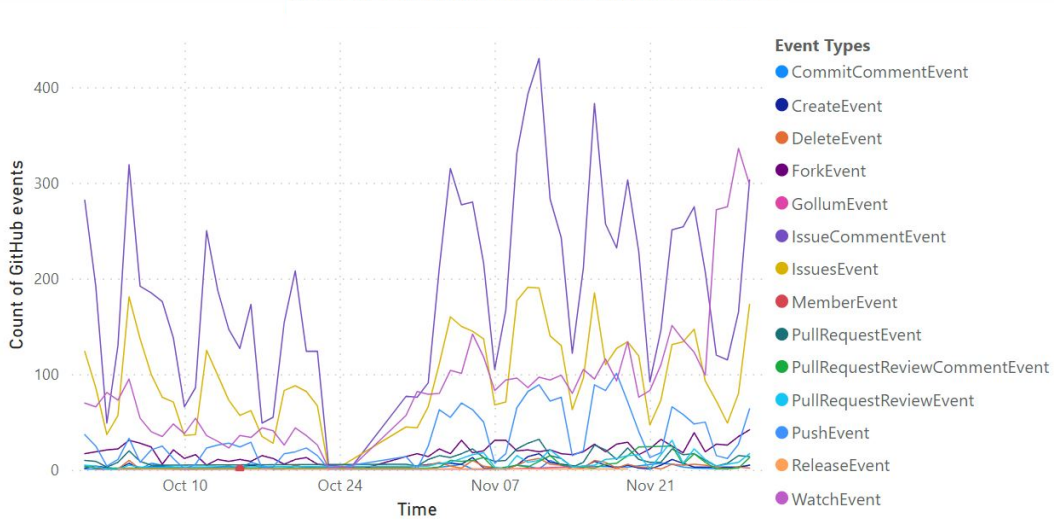

Figure 4.13: Counts of all daily GitHub events by Event Types October 2021

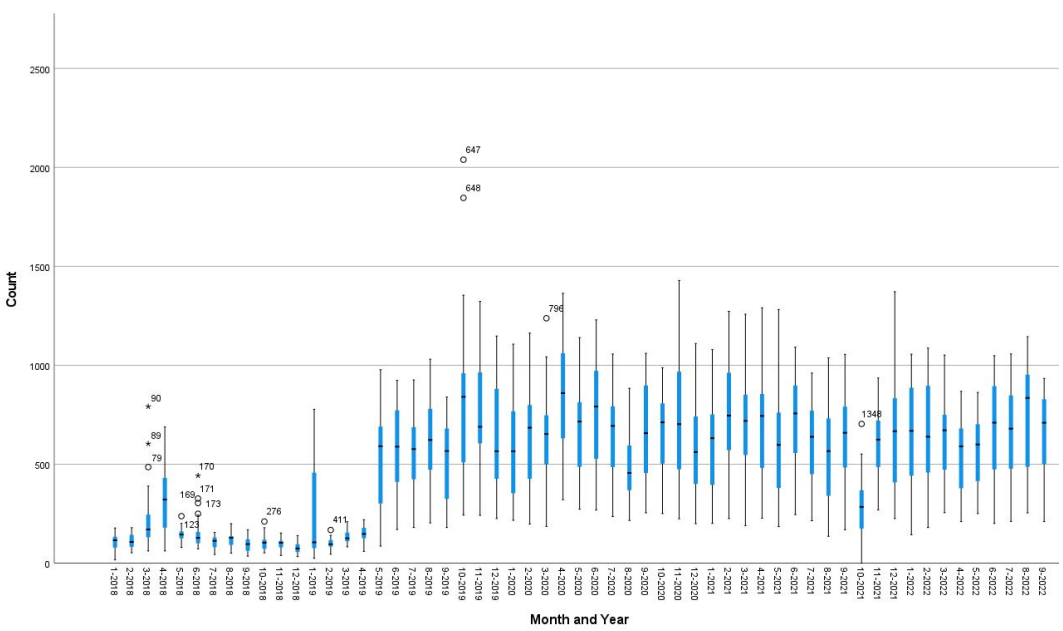

Figure 4.14: Boxplots Counts of all monthly GitHub events

showing no skewness this study accepts the missing data.

4.7.3 Relationships Between GitHub Events

In Section 4.4 the types of GitHub events present in the sample population dataset are analysed and the expected relationships between the events are listed in Section 4.4 and further detailed in Section 4.6. Statistical analysis is used to investigate the relationships between GitHub events present in the sample population dataset with the objective of testing the dataset for alignment to Figure 4.8. Investigation of the data set is carried out to determine whether previously unidentified relationships exist and identify trends in GitHub events detailed further in this section.

First, this study explores the relationships between the GitHub `event id`, event times, and the dates the events occurred by generating a 3D Scatterplot illustrated in Figure 4.15. The visualisation shows a relationship between `event ids` and event types, indicating that a change to a repository can have multiple GitHub event types associated with it. This supports the definition of `Event Id` described in Section 4.4.1, where we noted that an `Event Id` is a key that indicates a change that can have multiple GitHub event types against it. This study also notes that certain event types are not as frequently created against changes to repositories as others. This is investigated and further described below.

Figure 4.16 is a two-dimensional representation of Figure 4.15 where the `Created` data is removed to make visual cluster identification easier. Figure 4.16 shows that certain event types such as `Release`, `Pull` request review, `Member` and `Gollum` events are not as commonly present against changes as other GitHub event types. There also appears to be a set of GitHub events commonly appearing when repository changes occur. The scatterplot shows similar clustering patterns between `event id` and pull request and pull request review comments. The same clustering patterns appear for issue

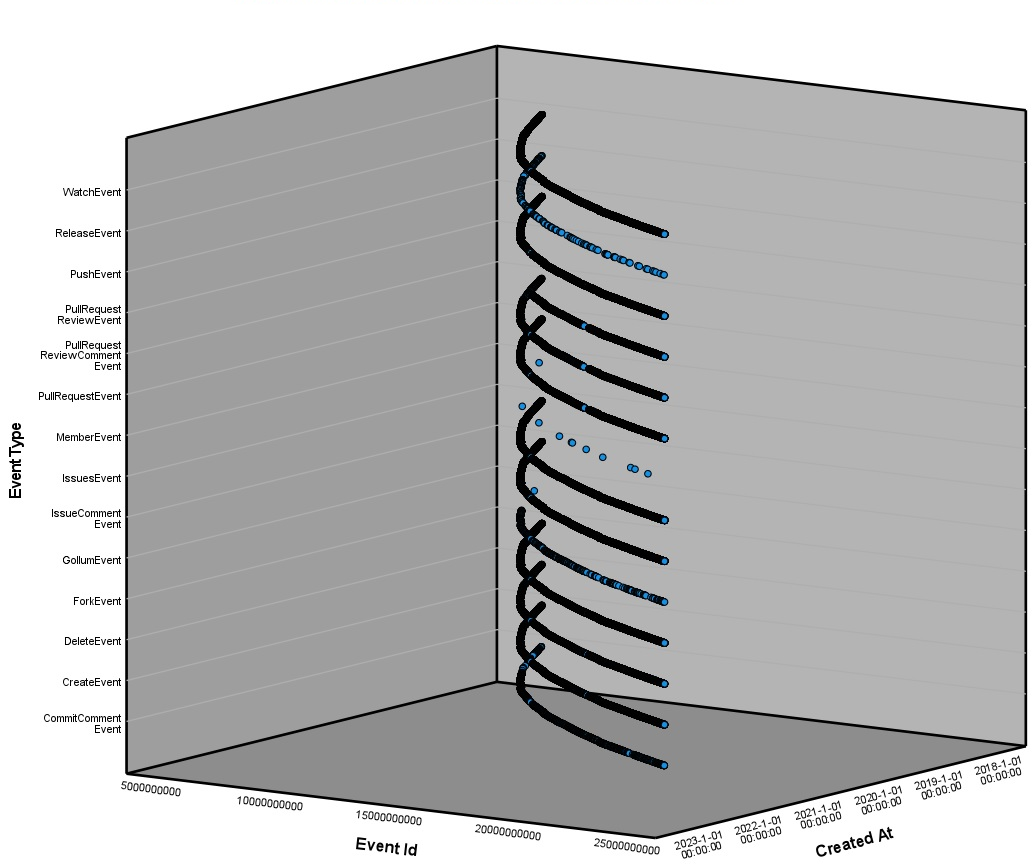

Figure 4.15: Event Types and associated to Event Ids by Created dates

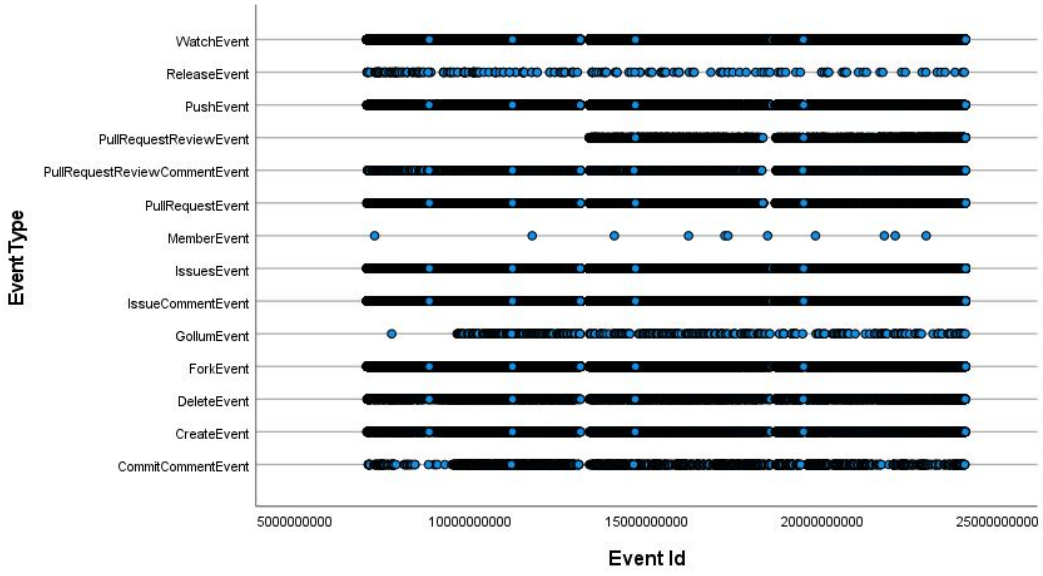

Figure 4.16: Event Types and associated to Event Ids

and issue comment events. This is further investigated through pairwise comparisons between GitHub events, analysed further in this section.

GitHub Events Relationships

To find relationships between GitHub events relevant to software enhancements, visual and statistical exploratory analysis of the sample population data set using Pairwise comparisons and Pearson Correlation statistics is carried out. Figure 4.17 illustrates a pairwise comparison of GitHub event types for the purposes of finding relationships between GitHub events. Figure 4.18 describes correlation statistics between GitHub events, such as Pearson correlation statistics that quantify the strength of a relationship between two variables. The pairwise comparisons and correlation statistics are conducted on a summarised GitHub events data set. The summarising activities are calculated by aggregating all GitHub events by day to get a count of each GitHub event count per day for the entire sample population data set.

A positive Pearson correlation value between 0 and 1 indicates a positive linear

relationship between two variables. This means if variable A and variable B are being compared, as variable A increases, variable B increases too. A correlation value between 0 and -1 indicates a negative linear relationship between two variables meaning if variable A increases, variable B decreases. A value of zero or close to zero indicates a weak or no linear relationship between variables. The categorisation of the strength of a correlation value is further detailed in Table 4.4.

Table 4.4: Pearsons correlation strength definition

Pearson Correlation Value	**Strength Definition**
±(0 to 0.2)	Very weak or no correlation
±(0.2 to 0.4)	Weak negative correlation
±(0.4 to 0.6)	Moderate negative correlation
±(0.6 to 0.8)	Strong negative correlation
±(0.8 to 1)	Very strong negative correlation

When evaluating the relationship between GitHub event types, events that are relevant to software changes are selected. `Gollum` and `Member` events indicate repository administration activities such as updating project documentation or managing project contributors. As such, these are not deemed relevant to the activity of software enhancements and are removed from pairwise comparisons. `Issue` comments, Commit comments and `Pull Request Review` comments are important to the validation of change requests or changes themselves but not the drivers of change to code and so are also removed from pairwise comparisons.

Pairwise comparisons illustrated in Figure 4.17 shows that overall there are positive relationships between GitHub events. The positive relationships are demonstrated by a positive trend line and dispersion of event counts around the trend line for GitHub event types. Examples of this is are the pairwise comparisons between watch and `Fork` events, `Push` and `Pull Request` review events and Create and `Delete` events.

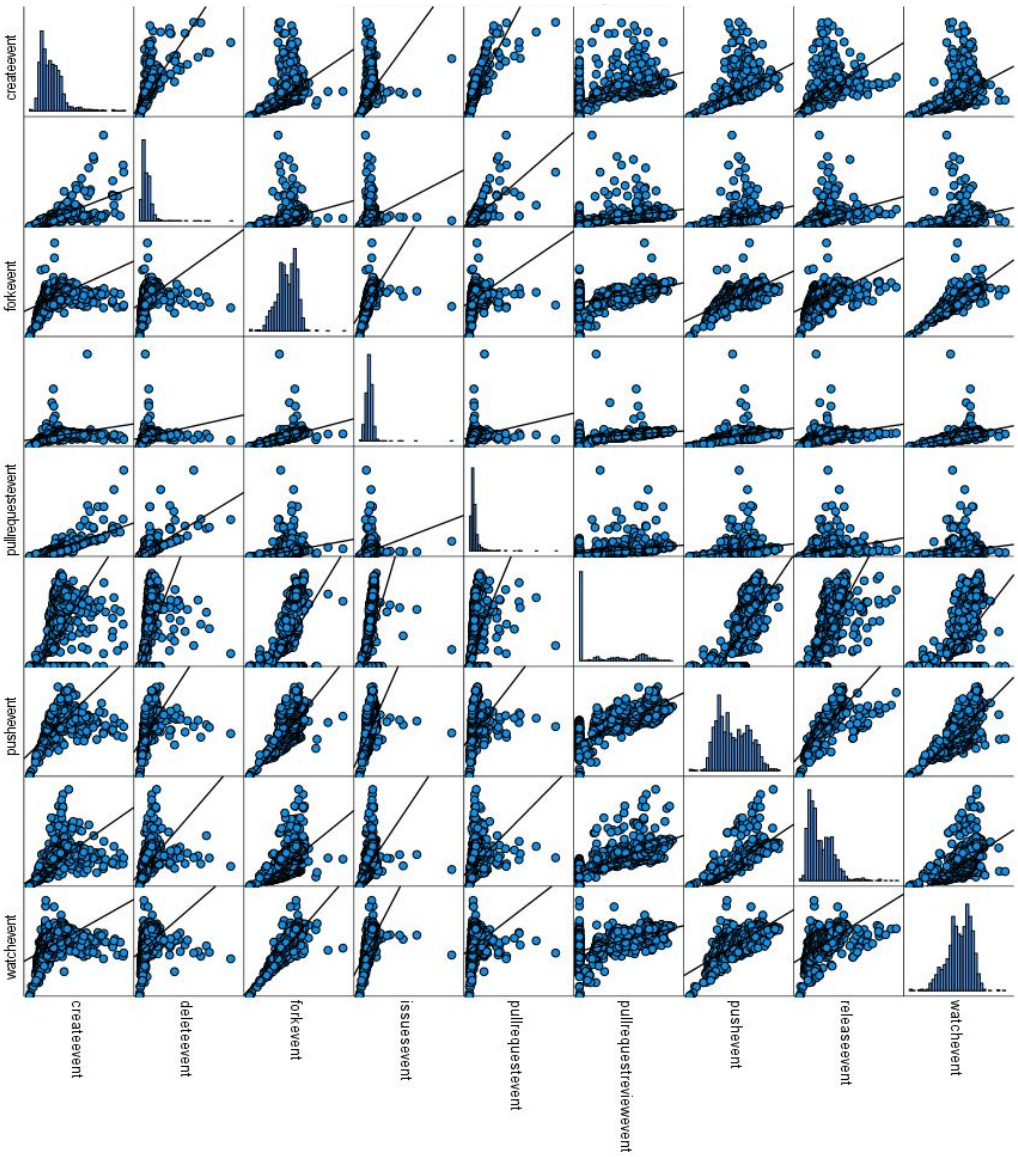

Figure 4.17: GitHub Event Type pairwise comparison

	createevent	deleteevent	forkevent	issuesevent	pullrequestevent	pullrequestreviewevent	pushevent	releaseevent	watchevent
createevent	1	.759**	.558**	.459**	.813**	.647**	.710**	.660**	.543**
deleteevent	.759**	1	.444**	.340**	.729**	.557**	.580**	.560**	.426**
forkevent	.558**	.444**	1	.652**	.353**	.565**	.762**	.633**	.910**
issuesevent	.459**	.340**	.652**	1	.300**	.476**	.564**	.472**	.617**
pullrequestevent	.813**	.729**	.353**	.300**	1	.427**	.408**	.408**	.316**
pullrequestreviewevent	.647**	.557**	.565**	.476**	.427**	1	.843**	.808**	.563**
pushevent	.710**	.580**	.762**	.564**	.408**	.843**	1	.862**	.782**
releaseevent	.660**	.560**	.633**	.472**	.408**	.808**	.862**	1	.636**
watchevent	.543**	.426**	.910**	.617**	.316**	.563**	.782**	.636**	1

0 to 0.2: Very weak or no correlation
0.2 to 0.4: Weak correlation
0.4 to 0.6: Moderate correlation
0.6 to 0.8: Strong correlation
0.8 to 1: Very strong correlation

Figure 4.18: GitHub Event Pearson Correlation statistics

The strength of the relationship varies between events as as dispersion is present an example of which is the relationship between create and release events. The strength of relationships between GitHub events is represented by Pearson correlations noted in Figure 4.18.

Pearson correlations show that there weak to very strong relationships between GitHub events with no very weak relationships existing. The results of Pearson correlations are summarised below in Section 4.7.3 below.

Weak correlations 11.11% of relationships between GitHub events show weak correlations. The lowest weak correlation exists between `Pull Request` and `Issue` events with a correlation value of 0.300 and the highest correlation value of 0.353 exists between `Pull Request` and `Fork` events.

Moderate correlations 44.44% of relationships between GitHub events show moderate correlations. The lowest moderate correlation exists between `Pull Request` Review and `Push` and `Release` events with correlation values of 0.408 and the highest correlation value of 0.584 exists between `Push` and `Delete` events.

Strong correlations 30.55% of relationships between GitHub events show strong correlations. The lowest strong correlation value of 0.617 exists between `Watch` and `Issue` events and the highest correlation value of 0.782 exists between `Watch` and `Push` events.

Very strong correlations 13.8% of relationships between GitHub events show very strong correlations. The lowest very strong correlation value of 0.808 exists between `Release` and `Pull Request` Review events and the highest correlation value of 0.910 exists between `Watch` and `Fork` events.

Investigation of Pearson correlation statistics as presented in Figure 4.18 indicates similarities to the ordering of the sequence of GitHub events as illustrated in Figure 4.8 and some minor differences. `Issues` events have strong correlations with `Fork` events

(0.652) and a moderate correlation strength with Create events (0.459). A relationship between `Issue` and `Watch` events (0.617) is also discovered and previously not identified in Figure 4.8. Pearson correlation statistics confirm the sequence of GitHub `Issue`, `Fork` and Create events and identified a previously unknown relationship between `Issue` and `Watch` events.

Figure 4.18 also identifies a strong correlation between Create and `Push` events (0.710) and `Fork` and `Push` events (0.762). This aligns with Figure 4.8, which indicated once a copy of the code base is made, developers would start pushing code changes to it. A very strong correlation between Create and `Pull Request` events (0.813) is identified and not noted as a direct relationship in Section 4.6. A very strong Pearson correlation between `Fork` and `Watch` GitHub events (0.910) is identified.

The correlation between `Push` and `Pull Request` events (0.408) indicating a moderate correlation. There are very strong correlations between `Push` and `Release` events (0.862) and `Push` and `Pull Request` Review events (0.843). These statistics indicate that there are likely to be more `Pull Request` Review and `Release` events triggered from a `Push` event, which is contrary to the process shown in Figure 4.8 where `Pull` requests are expected to result in `Pull` Requests being created.

`Pull Request` review events have a very strong correlation to `Push` events (0.843) and `Release` events (0.808). This aligns with Figure 4.8 where a `Pull Request` review can be rejected, requiring more changes to be made, which are represented by `Push` events or changes being accepted and released, resulting in `Release` events.

The sequence of GitHub events illustrated in Figure 4.8 does not fully align with the correlations and pairwise comparisons described in this section. For example, `Push` events are expected to create `Pull Requests`; however, correlations are stronger between `Push` and `Pull Request Review` events and `Push` and `Release` events. `Watch` events feature strongly in the SDLC and have strong relationships with `Issue` Events (0.617), `Release` events (0.636), `Push` events (0.782)

and a very strong correlation with `Fork` events (0.910). `Watch` events aren't described as an activity which is part of the software delivery activities in Figure 4.8 but are be investigated and discussed further in Chapter 5.

Pearsons correlations and the sequence of GitHub events, as noted in Figure 4.8, indicate that `Issue` results in code changes which are committed by `Push` events. `Push` events eventuate into software releases through `Release` events. These relationships may be indirect, such as the case of `Issues` to `Push` events which are typically intermediated by Create or `Fork` events. `Issue` and `Push` events are used in Section 4.8 to create a predictive model.

4.8 Predictive Model Development

In this section, the process of creating a predictive model forecasting the issues logged by software users resulting in software development efforts and releases made by software delivery teams is discussed. This is done by creating an SVARIMA time series model as detailed in Section 3.6.4. The variables used in creating an SVARIMA model are GitHub Issue and `Push` events, as discussed in Section 4.7.3.

4.8.1 Dataset Definition

In this section, a subset of GitHub events data from the total population is analysed and used to create a predictive SVARIMA model. The sample cases, as in previous sections of this chapter, are listed in Section 3.6.3. The data set used in this section contains records from May 2019 to September 2022 and is slightly over three years worth of sample data. The sample data is split into three subsets of data which is used in the training and validation of an SVARIMA model. The breakdown of sample data sets is listed in Section 4.8.1.

Full dataset All GitHub events for the selected cases from May 2019 to September 2022. This dataset is used for statistical exploration and analysis of data for skewness and stationarity. This dataset is corrected as needed based on the results of statistical exploration id est, skewness.

Training data set All statistically corrected GitHub events from May 2019 to June 2022 taken from the full data set. This data set is used in training an SVARIMA model.

Testing dataset All statistically corrected GitHub events from June 2022 to September 2022. The testing dataset is a reserved set of data from the sample dataset not used to train a forecasting model with. This dataset is used to validate forecasts from SVARIMA models with actual data observed for that time period. This dataset contains four months of data.

4.8.2 Skewness Tests

To create SVARIMA models, data needs to be normally distributed. In Section 4.7.1 all GitHub events are summarised and tested for skewness with no corrections being identified. Individual GitHub events are now analysed for skewness using the full data set noted in Section 4.7.3. The results, shown in Table 4.5, indicate that Issue events are highly skewed to the right with a skewness value of **1.8166** and requires correction. `Push` events show near symmetrical skewness, **0.4493**, requiring no further correction. The statistical skewness values are supported by the pairwise comparison distribution plots illustrated by Figure 4.17.

Yeo-Johnson transformation is used to correct skewness for Issue events. Yeo-Johnson is used due to there being zero values in the dataset which means Box-Cox transformations could not be used. Log and Cube Root transformations are outperformed by Yeo-Johnson and so not used. The transformation methods applied are noted

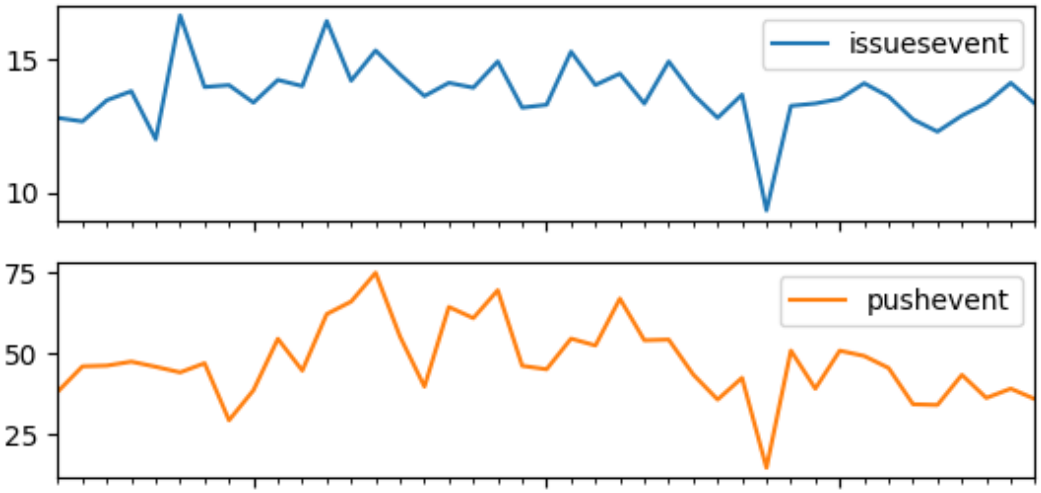

Figure 4.19: GitHub Issue and Push event Correlation monthly time series

in the Transform Method column in Table 4.5 and the new skewness values of the corrected data in the Corrected Skewness column.

Table 4.5: GitHub events variable skewness

GitHub event	Skewness	Transform Method	Corrected Skewness
issuesevent	1.8166	Yeo Johnson	0.0931
pushevent	0.4493	None	0.4493

With skewness corrected, the temporal characteristics of variables are evaluated. The daily Issue and `Push` events contained in the full dataset are re-sampled to get a monthly mean and plotted on a time series graph shown in Figure 4.19. Visual examination of Figure 4.19 indicates data also exhibits similar temporal characteristics to those described in Section 4.7.2 where data is not missing values and there are no significant outliers in the dataset. Trends in the data also indicate there is a monthly seasonality of data.

4.8.3 Stationarity Tests

Stationarity of data is important for time series forecasting as statistical analysis requiresthe data set is consistent over time. Stationarity of data is tested using ADF tests as described in Section 3.6.4. ADF tests for `Issues` events return a p-value of 0.0154 and test statistic of -3.2865. `Push` events have a p-value of 0.1132 and test statistic of -2.5090. The ADF results for Issue events indicate the null hypothesis of the data being non-stationary is rejected. The ADF results for `Push` events indicate the null hypothesis of the data being non-stationary is accepted.

SVARIMA models require data to be stationary and as a result of the previously mentioned ADF test results, `Push` events need to be differenced. One order of differencing applied to `Push` events returning an ADF p-value of 2.1627×10^{17} and test statistic of -9.9771. The results indicate that `Push` events are mate stationary with one order of differencing applied.

With both `Push` and Issue events distributed normally and confirmed as being stationary, these values can be used as variables in creating SVARIMA models. The process of creating, testing and validating a model is described in Section 4.8.4.

4.8.4 SVARIMA Modelling

In this section the process of creating SVARIMA forecasting models for GitHub Issue and `Push` events and their outputs are discussed. When creating the SVARIMA models, the Python auto ARIMA library is used. Auto ARIMA is able to test a model against different parameters to find the best fit, which is measured by the AIC goodness of fit test result. The model creation, parameter selection and training process and the model testing and validation processes are discussed in the following sub sections.

```
model_pushevent = pm.auto_arima(final_df['pushevent'],
                        m=12, seasonal=True,
                      start_p=0, start_q=0, max_order=maxorder
                         , test='adf',error_action='ignore',
                           suppress_warnings=True,
                      stepwise=False, trace=True)

fit_results_pushevent = model_pushevent.fit(train['pushevent'
   ])

model_issueevent = pm.auto_arima(final_df['issuesevent'],
                        m=12, seasonal=True,
                      start_p=0, start_q=0, max_order=maxorder
                         , test='adf',error_action='ignore',
                           suppress_warnings=True,
                      stepwise=False, trace=True)

fit_results_issueevent = model_issueevent.fit(train['
   issuesevent'])
```

Listing 4.2: Auto ARIMA model creation

SVARIMA Model Selection and Training

In this section, the process of creating and training a forecast model is detailed. To create SVARIMA forecast models for `Push` and Issue GitHub events, the optimal parameters for the model need to be found. This is done using auto ARIMA libraries and is known as model creation and training. The auto ARIMA script in Section 4.8.4 tests multiple *(p,d,q)* and *(P,D,Q)s* permutations against AIC values returned by goodness of fit tests using the training dataset previously mentioned in Section 4.8.1. The model parameter combination with lowest AIC value is returned as the optimal model. The auto ARIMA configurations are further described in Section 4.8.4.

final_df specifies the variable used to create a model, in this case `Push` and Issue events. The variables used in this model contained all data for the selected cases from May 2019 to September 2022. This dataset is described as the full dataset in Section 4.8.1. All data contained in this dataset are normally distributed as

discussed in Section 4.8.2 and are stationary, as discussed in Section 4.8.3.

m = 12 m is the seasonal periodicity of the time series data known as seasonality. The seasonality of this dataset is identified in this chapter to be monthly and so is set to 12.

seasonal = True The seasonal=True configuration specifies that the model generated needs to cater for seasonality.

max_order = maxorder max_order specifies the maximum AR and MA components for this model. This is specified in a configuration variable set at 50 as experimentation shows this yielded the optimal fit for the least compute time used.

test = adf Test = adf specifies that the model parameter selection process needs to ensure data stationarity is tested using the ADF test and the model differences data if required.

The code in Section 4.8.4 generates two models, a `Push` events and Issue events forecasting models. The model for `Push` events is (3,0,0)x(0,1,0)12 with an AIC of 205.580 and BIC of 211.675. The model for Issue events is (0,1,1)x(2,0,0)12 with an AIC of 114.244 and BIC of 122.161. Auto ARIMA applied seasonal differencing to the `Push` events model and non seasonal differencing for the `Issues` model. ADF tests specified in Section 4.8.3 indicate the data is stationary after differencing is applied however Auto ARIMA has found that parameter tuning yielded better (lower) AIC results for models through applying seasonal and non seasonal differencing, as specified by the models generated.

In this paragraph the forecast model residual statistics recorded in Table 4.6 are discussed. The `Push` events model has a Ljung-Box p-value of 0.02 and Issue model has a Ljung-Box p-value of 0.24 meaning the null hypothesis of model residuals not having any autocorrelations is rejected. This means the residuals of the predictive

models are not random, but have a relationship with time thus making it suitable to time series analysis (Dabral & Murry, 2017). The models also show that residuals are normally distributed with the `Push` model residuals having a skewness of -0.01. The `Issues` has a residual skewness value of 0.58 meaning the data is slightly right skewed. Further investigation is needed to determine additional skewness correction of data, however due to time resource constraints, this is determined to be out of scope of this study.

Table 4.6: Forecast model residuals statistics

Model	Ljung-Box	Skewness	AIC	BIC
`Issues` (0,1,1)x(2,0,0)12	0.24	0.58	114.244	122.161
`Push` (1,0,1)x(2,1,0)12	0.02	0.01	205.580	211.675

With the forecast models created, trained and understood, the testing and validation of the SVARIMA models are carried out The testing and validation processes are further discussed in the next Section.

4.9 Predictive Model Testing

In this section, the process of testing SVARIMA models created in Section 4.8.4 to generate forecasted values or an 'unseen' date range projecting from the end of the training dataset date range to the end of the validation dataset date range, which is four months for this study. The results of forecasts are then compared to the testing dataset containing observed values for the same time period as the forecasts and illustrated by Figure 4.20.

4.9.1 Push Event Forecast Tests

The `Push` events forecast model results follow similar trends to observed data. For example, the `Push` events forecast model, as illustrated in Figure 4.20 and Figure 4.21, shows that the number of monthly `Push` events count is lower at the end of quarter two calendar year when compared to the middle of that quarter for all quarter two forecasts. This is supported by the trend observed in historical data where the `Push` event counts are lower at the end of quarter two when compared to the middle of the quarter from 2019 to 2022.

The same observations can be applied to quarter three where more events occur in the middle of the quarter compared to the end of the quarter for observed and forecasted values. Quarter four shows that more `Push` events are observed at the end of the quarter compared to the middle of the quarter and is matched with forecasted values.

`Push` events observed for quarter one for 2020 and 2021 show less `Push` events recorded at the middle of the quarter compared to the end of the quarter. This does not match the observed values for 2022 where there are more `Push` events in the middle of the quarter compared to the end of the quarter. The forecast `Push` events model is similar to the observed values for 2022 where forecasted values show that more push events are forecasted for the middle of quarter one compared to the end of the quarter. This is likely to be due to quarter one observed values influencing the forecasting model.

4.9.2 Issue Event Forecast Tests

The `Issues` events forecasts trends also match the observed trends in historical data. Observed quarter two trends for 2019 to 2022 show that the number of `Issues` observed at the beginning of quarter two drops towards the middle of the quarter and increases from the middle to the end of the quarter. The Issue event forecast model matches this.

Quarter three observed values for 2019 and 2020 show an increase in `Issues` events from the middle to the end of the quarter. The trend for 2021 and 2022 shows the number of `Issues` events observed in quarter three drops from the middle of the quarter to the end of the quarter, which is the opposite trend from that observed in 2019 and 2020. The Issue events forecast model for quarter three 2022, shows a similar trend to those observed for quarter three 2022 2021 and 2022, matching observed values; however, the quarter three trend for 2023 and 2024 aligns with those observed in 2019 ans 2020. The Issue event forecast model also shows a steady projected increase in expected `Issues` from quarter four 2022 onward.

4.9.3 Push and Issue Event Forecasted and Observed Dataset Comparison

In this section, forecasted and test data is described and compared. Table 4.7 contains details of the observed, forecasted and confidence interval values of both models. The results are also illustrated in Figure 4.20.

Forecasted values for `Push` events don't align closely to observed counts but do align with the trends discussed earlier in Section 4.9.1. The differences in forecasted and projected event counts could be due to the SVARIMA model factoring in the trends in 2020 and 2021 where the dips in monthly `Push` event counts are lower than those demonstrated in 2022, and so the model likely factors in that observed historical data and trends, lowering the forecasted monthly counts to match historical values. As such, the forecasted `Push` events trend does follow historical trends but with lower forecasted values predicted.

Forecasted `Issues` events counts are slightly lower than observed values; however, the deviation is smaller than those forecasted by the `Push` events model. The confidence interval for `Issues` forecasts is also narrower than those of the `Push` events model,

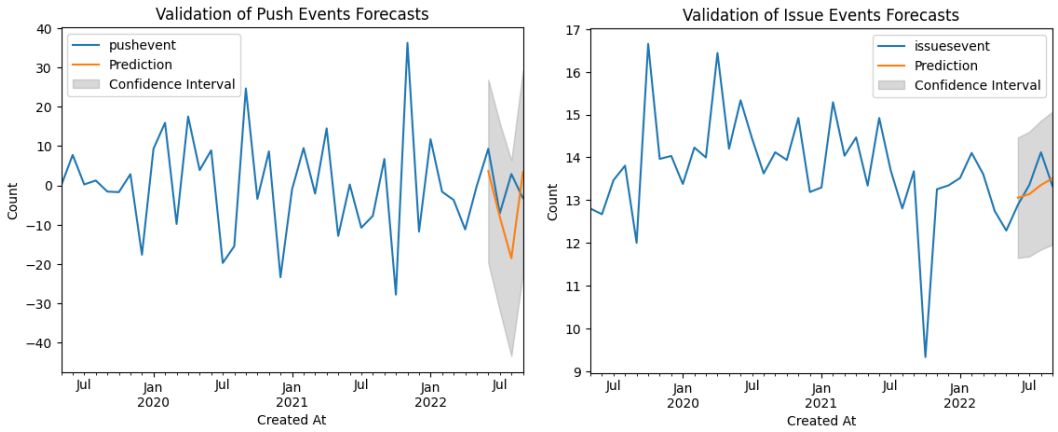

Figure 4.20: SVARIMA 4 month forecast tests for Push and Issue GitHub events

meaning the `Issues` forecasts have a higher degree of certainty. This indicates a better fit for the Issue, which is also indicated by the lower AIC value of $\mathbf{114.244}$ of the Issue events model compared to a $\mathbf{205.580}$ AIC value for the `Push` events model.

Table 4.7: Forecast validation results for GitHub events

Model	Date	Forecast	Lower CI	Upper CI	Actual
`Push`	2022-06-30	3.6194	-19.6815	26.9204	9.3000
`Push`	2022-07-31	-8.0267	-31.9392	15.8856	-7.1387
`Push`	2022-08-31	-18.5419	-2.3561	6.3510	2.8709
`Push`	2022-09-30	3.3481	-10.9082	29.1554	-3.2544
`Issues`	2022-06-30	13.0574	11.6503	14.4645	12.8889
`Issues`	2022-07-31	13.1422	11.6845	14.5999	13.3588
`Issues`	2022-08-31	13.3531	11.8464	14.8597	14.1189
`Issues`	2022-09-30	13.5117	11.9577	15.0657	13.3298

4.9.4 Long Term Forecast Examination

In this section, the long term forecasts produced by the `Push` and `Issues` events forecast models are examined. Examination of long term forecasts involves generating a

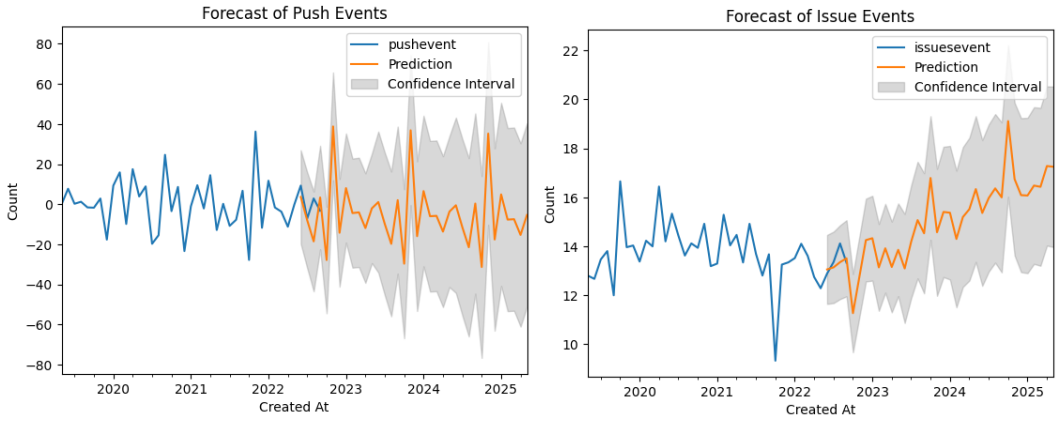

Figure 4.21: SVARIMA 36 month forecast for Push and Issue GitHub events

36 month forecast for `Push` and `Issues` events and plotting observed and forecasted values for visual examination. The forecast trends are then examined visually to compare against observed trends for similarities or differences.

The results of a 36 month forecast of the `Push` and Issue event models and observed data are illustrated in Figure 4.21. Examination of `Push` events forecasts show that the quarterly trends from quarter two 2022 to quarter two 2025 are similar in characteristics to trends identified in Section 4.9.1. Issue events forecasts from two 2022 to quarter two 2025 also exhibit the similar trends as those described in Section 4.9.2. It is noted that the long term forecasts for Issue events predict an increase in the number of expected `Issues` being created from 2023 to 2025. Although the seasonality of forecasts are similar to historical data, the number of issues occurring shows a consistent increase. This is further discussed in Chapter 5.

With the examination of long terms forecasts complete, it is determined that both the `Push` and `Issues` events forecast models are suitable for further investigation. In Section 4.9.5 the forecast models created are applied to observed software releases for the sample cases to assist in answering the research questions discussed in the following section.

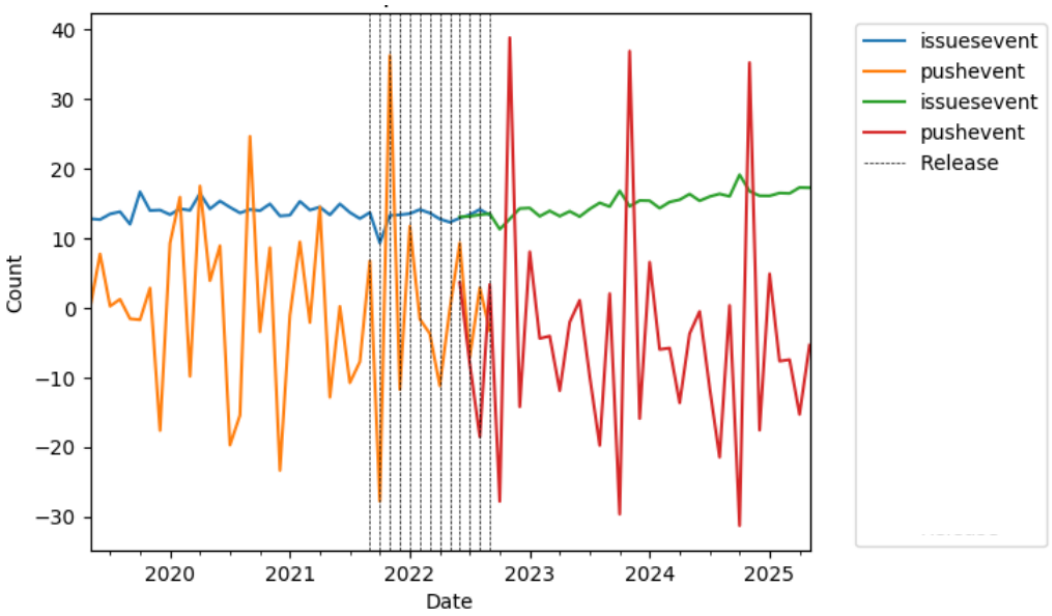

Figure 4.22: SVARIMA 36 month forecast and actual values for Push and Issue GitHub events with releases

4.9.5 Relationships Between Push, Issue and Release Events

The research questions in Section 2.7.1 seek to find a way to identify and select software enhancements that add value, identify how value is determined and if an evidence-based method could be used to do the previously mentioned things. As such, the release of software is important in answering these questions as they represent the delivery of value. The relationship between `Issues`, which are requests for change, software development activities resulting in code changes, represented by `Push` events, and the delivery of changes, represented as `Release` events, are examined in this section.

Figure 4.22 illustrates the combined historical and forecasted data for `Push` and Issue events with GitHub `Release` events added. The observed data used, as per other sections of this chapter, represents the selected sample cases for this study specified in Section 3.6.3. Observed software releases for the selected cases are shown as vertical black dash lines.

Figure 4.22 shows a relationship between observed `Push` and Issue events where

an increase in the number of Issue events has a corresponding increase in the number of `Push` events before releases occur as demonstrated where releases are illustrated in Figure 4.22. After releases occur, the number of `Push` and Issue events logged per month show a reduction in volume. Examples of this trend are shown in quarter three 2021 and quarter one 2022. If a reduction in `Push` and `Issues` events do not occur, the rates of growth for both events slows as shown in quarter 4 2021.

Figure 4.22 also shows that the peak in `Push` event counts for a month usually results in a release occurring. Forecasted values for the validation period also follows these trends when compared to the observed events. These observations are further discussed in Chapter 5.

In this section Software releases are compared to observed and forecasted `Issues` and `Push` events to find relationships between them. Analysis of observed and forecasted values show that release events do have an impact on `Push` and Issue events. The visual examination of Figure 4.22 carried out in this section is supported by the Pearsons correlations statistics in Figure 4.18 where a relationship between these three variables was identified.

With the predictive model for `Push` and `Issues` events created, tested and queried against `Release` events, it is found that the model is able to deliver predictive insights. To ensure the model can be generalised, validation of the forecast model needs to occur. Section 4.10 describes the validation of the `Push` and `Issues` event models created in this section.

4.10 Predictive Model Validation

In this section the process of validating the SVARIMA forecast model created and tested in this chapter is discussed. The validation process steps carried out in this section represent the artefact evaluation step of the DSR process. To carry out forecast model

validation, the Push and Issue events forecast models created in this chapter are tested against a new case to see if the predictions and trends of forecasts align with previously unseen data points.

To validate the forecast model, a GitHub project not used for training or testing the forecast model is compared to the forecasted values of the Push and Issue event models. For the purposes of this study, the Brave web browser is selected as the validation project. This is due to it being a cross-platform user facing application which is actively being worked on and having events that match the time period of sample cases. This data set is known as the validation data set.

Before comparing observed values for Brave events to forecasted events, the skewness and stationarity of the validation data set is carried out and corrections are applied. The pre-corrected skewness of Issue events are 0.5593, and Push events 2.8745. Yeo-Johnson transformation is applied to both variables resulting in skewness of Issue events being corrected to -0.1388 and Push events are corrected to -0.0181. Stationarity tests for variables resulted in an ADF value 0.0154 for Issue events and 0.1132 for Push events. Push events are differenced, resulting in an ADF value of 2.1627×10^{-17}.

With skewness and stationarity corrected, observed and forecasted values, along with release events, are plotted. This is illustrated in Figure 4.23. Figure 4.23 shows that the forecasted values are larger when compared to observed values and that trends for Push and Issue events for the validation data set are the same as those observed in the model testing process described in Section 4.9.

Figure 4.23 also shows a relationship between the number of Push and Issues events are shown where an increase in Issues events has corresponding increases in Push events before releases occur. This is demonstrated by the increase in Push and Issues events leading up to a release in September 2021 and November 2021. The number of Push and Issue events decrease after releases, as demonstrated by the decrease in events after the September 2021, November 2021, April 2022, June 2022 and

August 2022 releases. There are examples of the number of `Issues` increasing after a release, as demonstrated by the October 2021 and November 2021 releases. There are also decreases in the number of `Push` events after releases shown for September and November 2021. The trends observed for the validation data set are very similar to those observed in Section 4.9.5.

The trends observed for forecasted values also match the observed values for the validation data set although at a higher level where the monthly count of activities forecasted for `Push` and `Issues` events is far larger than those observed. This is the result of the number of activities against the sample cases being higher than the validation case and is further discussed in chapter 5. The forecast model created is found to be generalisable as it exhibits the same characteristics and behaviours as the training, testing and validation cases.

In Chapter 5 the findings from this chapter are further discussed and related back to the research questions specified in Section 2.7.1. Chapter 5 also discusses the testing and results of study hypothesis specified in Section 3.5.2 against the findings of this chapter.

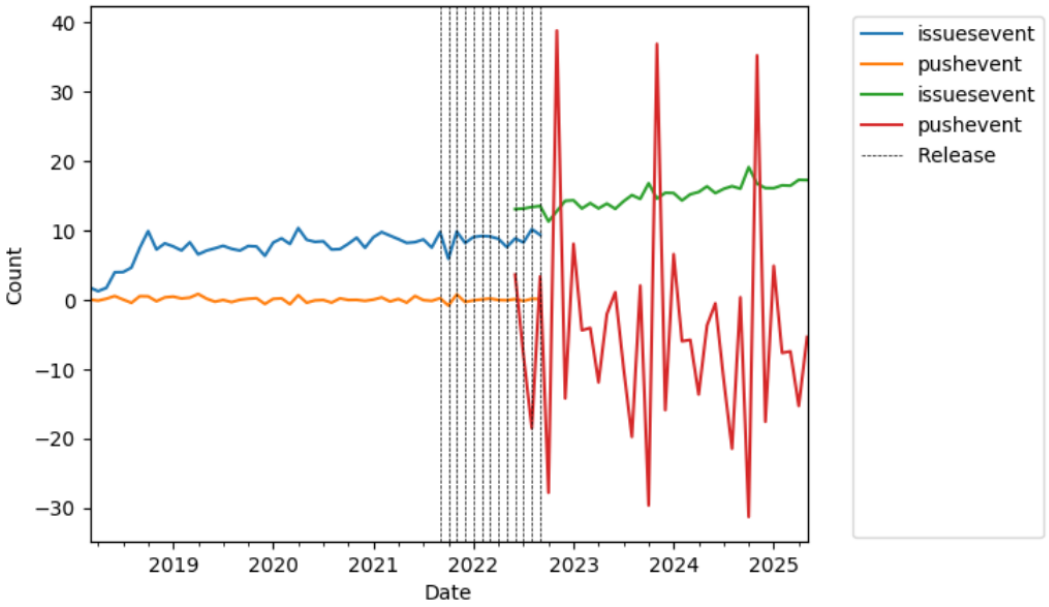

Figure 4.23: Brave browser observed and 36 month forecasted values with releases

Chapter 5

Discussion

5.1 Introduction

This chapter discusses the observations made from carrying out the analysis detailed in Chapter 4. The chapter also answers the research questions and addresses whether the objectives of this study are met. This chapter also addresses the research hypothesis by testing them against the results of the analysis executed in this study. The major findings from the analysis carried out in Chapter 4 shows that a relationship between `Push` and `Issues` events exists that is detailed in Section 5.2 indicating that an evidence-based approach to selecting software enhancements can be taken. This is further discussed in Section5.3.

5.2 More Push Events result in fewer Issues

The results obtained from the SVARIMA model in Section 4.9 and Section 4.10, indicate that an increase in the number of `Push` events associated with software releases leads to a decrease in the number of post-release `Issues` logged. During months with a higher count of `Push` events, the total number of `Issues` logged after the release is

lower when compared to months with fewer `Push` events. This pattern is illustrated in Figure 4.22 for the September 2021 and March 2022 releases. In these cases, software releases with fewer `Push` events in the preceding months result in higher counts of `Issues` logged, as opposed to the months of August and October 2021 which have a higher number of `Push` events resulting in fewer post-release `Issues` being logged.

Additionally, the predictive model's validation case, depicted in Figure 4.23, displays a similar trend. For example, releases from September and November 2021 show a decreased count of post-release `Issues` logged in comparison to the preceding month, October 2021, and the subsequent month, December 2021. October and December 2021 have fewer `Push` events than September and November 2021 and as such, the count of post-release `Issues` logged demonstrated an increase in these months.

The number of `Push` events within a repository can be used as a metric for gauging software developer effort or the number of changes applied to a software release. As such, an increase in the count of `Push` events can correspond to a more substantial implementation of features or resolution of bugs. Therefore, the observed connection between releases with more `Push` events encountering fewer post-release `Issues` could be due to releases addressing existing bugs or fulfilling functionality requests previously logged as `Issues`. This can result in the closure of a large number of reported `Issues`. The feature or bug fix released could also be a frequently reported Issue and so the release may stop these duplicates from being logged, thereby reducing the overall count of `Issues` after the release.

`Push` events can trigger code reviews and frequent smaller changes are easier to review for quality issues. Releases with a higher number of `Push` events can indicate smaller incremental software changes committed frequently. As a result, applying quality reviews to code changes is easier and could result in bugs being picked up more frequently as the cognitive complexity of each smaller `Push` event is lower than their larger counterpart.

It is important to note that a reduction in the number of post-release `Issues` logged may not be a sign of software user satisfaction. Releases can introduce bugs and feature changes that negatively impact software users. As such releases run the risk of software users abandoning an application due to dissatisfaction. Software application users may also abandon an application due to software delivery teams releasing changes that are not important to software users.

Frequent code commits can be used to create frequent releases for software applications. With the ability to release software changes at a higher frequency, the latency between a release and a software publisher's ability to measure the release's impact on customers reduces. Releasing more frequently and measuring the impact of the change on customers using a model like the one created in this study can allow software publishers to use an evidence-based approach in identifying the impact of releases on customers and experiment with the type of releases prioritised.

5.3 Evidence Based Software Enhancements

The SVARIMA model created in Chapter 4 can be used to predict future application user feedback trends. This study categorises bugs and feature requests as `Issues`. However, a future iteration of the model could be made to predict what types of user requests a software delivery team could deal with. A forecast of bugs and feature request numbers can assist software delivery teams in their workforce planning and identifying areas of improvement in their software delivery processes to improve customer satisfaction. For example, if the forecast illustrates an increasing number of bugs, the quality assurance processes of a software delivery team can be adjusted to ensure releases do not negatively impact customer satisfaction or market perception of an application.

Another scenario of using evidence to improve software delivery is prioritising and selecting features to be released. An extended version SVARIMA model produced in

this study, where bugs and features can be independently forecasted, can be used to test the impact of feature releases on software users. The literature review indicates that feature selection decisions are often made by software Product Management teams using intuition, as described in Section 2.4. The forecast model produced in this study can be used to measure the impact of feature releases on customers using an evidence-based approach. As such product teams can create and test hypotheses on feature selection for releases by experimenting on the prioritisation of software enhancements contained in a release and measuring the impact of the release.

Carrying out experiments in software delivery process changes and measuring their impact on forecasted `Issues` can be done by using a model similar to the one produced in this study, allowing an evidence-based approach to selecting software enhancements. This can assist in identifying the impact of changes in software delivery processes to customers and also help avoid software abandonment by the users of applications described in Section 5.2.

Tools can be developed to assist in prioritising, selecting and delivering features or measuring the impact of software delivery process changes mentioned above. The forecast model produced in this study can be integrated with software delivery work management tools, such as Azure DevOps, GitHub or JIRA, used by software delivery teams and users to identify bugs, record and manage features and manage source code (Shrivastava & Shrivastava, 2024). Such an integration can provide an ongoing stream of input data to the forecast model, resulting in predictions of the impact of software releases occurring continually. The forecasted results can be captured in a report tool such as a business intelligence dashboard. A dashboard can be used by software delivery teams in their work planning or process improvement processes to identify features to be prioritised or the impact of changes made to software development practices. Such a tool can also be used to validate the SVARIMA forecast model produced in this study, as real world data can be used to predict the projections made by the model.

5.4 Evidence Based Software Selection

The model produced in this research can be used not only by publishers of software applications but also consumers. As individuals and organisations adopt software, the quality of the application is identified as key to the selection process. The literature review carried out in this research has identified quality assessment frameworks discussed in Section 2.6. Software uses can use the SVARIMA model produced in this research to assess the effectiveness of releases published to identify the impact of historical and forecasted releases.

With enhancements applied in future iterations of this study, the `Issues` could be further categorised into features and bugs relating to the assessment criteria of the quality frameworks relevant to assessors, for example reliability, efficiency, etc. Using an evidence-based approach to identifying the maintainability of software and understanding the behaviours of software publishers, the total cost of ownership of an application can be better understood by software users. The use of evidence-based software selection can be further studied in another evolution of this research as it builds upon the model created in this research.

5.5 Research Questions

The research questions for this study are listed in Section 2.7.1 and answered in this section. RQ1 asks how can a software development team select software updates that adds value with confidence. The findings of this study indicate that software user feedback provided in the form of `Issues` logged can be used to identify software updates that add value. Identifying key themes derived from `Issues` logged, addressing them and measuring feedback as described in Section 5.3 helps software delivery teams determine the impact of releases and infer value to users. Further modifications to the model

produced in this study to add variables such as software downloads and GitHub stars against projects can be applied to identify software's historical and forecasted popularity and the changes after releases occur.

RQ2 asked what are the factors that affect or determine value. The literature review identifies quality as a key driver in determining the value of software. Measures of quality can be categorised into criteria that attempt to measure the ability of software to solve problems for users, the resilience and performance of software et cetera. Measures of quality and the quality assessment frameworks used to measure them are discussed in Section 5.4. Quality metrics specified in various quality assessment frameworks can be measured using evidence-based methods such as literature reviews of products or analysis of software delivery processes identifying metrics such as features selected for releases, effort applied to remediating bugs and the number of issues created after releases.

RQ3 asks how an evidence-based approach can be applied to software enhancements. The use of evidence in identifying enhancements are discussed in Section 5.3 and the ways in which the effectiveness of enhancements can be measured is discussed in this chapter. The forecasting model produced in this research can be used to select software enhancements and measure the impact of releases containing selected enhancements. Methods discussed in this chapter discuss how experiments can be executed within the software delivery process to select different types of enhancements and measure the impact to users.

5.6 Limitations

In this section, the limitations of the analysis are highlighted. This study only uses Open source projects which results in a subset of software titles and subsequent development activities being analysed. This results in Closed source applications not being captured.

This may limit the ability to generalise the model created as model creation and validation against closed source applications have not occurred. The data source for this study is GitHub which is an extremely popular code repository, but other sources of software development activities, such as Azure DevOps or Team City are not used. In future studies, closed source projects and other data sources could be used to enhance this study so results can be generalised to open and closed source software delivery teams.

`Issues` can be created for bug reporting, feature requests or general discussions. The analysis carried out in this chapter does not distinguish between those event types due to data quality issues of GitHub events. These quality issues are due to `Issues` not being consistently tagged as Bugs or Features resulting in difficulty in distinguishing between those event types safely. The risk of this limitation is mitigated as `Issues` on GitHub are created for the purposes of Bug reporting and Feature requests meaning the events are very likely Bugs and Features. This study also ignores Issue comment events to filter GitHub events that are community chatter.

In future studies, it is recommended that more time is spent in classifying Issue events as Bugs or Features and ensure that only these types of `Issues` are used in the creation of forecast models. If `Issues` events are classified as Bugs or Features, forecast models for each of these events could be made to investigate the influence Bugs or Features further separately have on software releases. Due to time constraints, the Issue classification could not be done in this study.

The selected cases are software applications a technical community uses, such as Software Developers or IT professionals. This introduces potential bias because a subset of software application users are creators of Bug reports or Feature request `Issues`. The use of a web browser to validate the forecast model shows this bias may not be pronounced; however, in future studies, applications used by non-technical users could be used to create forecast models that are compared to the models created in this study to measure differences in results.

Finally, this study does not analyse the negative impact of frequent software releases on users or software delivery teams. As valuable software changes are identified and prioritised, software delivery teams may become pressured to increase the cadence of releases. There may be possible negative impacts of this, as fatigue may cause developer burnout, attrition, or mistakes resulting in an increased number `Bugs` in releases. The impact on software users needs to be investigated, too, as more frequent releases may result in frustrations due to user interface changes and updates requiring productivity time being spent on testing leading to resource contention. Surveys and experiments could be designed to capture human feedback and the impact of delivery cadence changes within software delivery teams. User satisfaction surveys could also be used to capture the impact on software users.

5.7 Conclusion

This study finds that the quality of applications determines the value of software to its users. Quality is measured using various quality frameworks from software delivery teams and users as discussed in Section 2.6. Software delivery teams can use quality frameworks to assist in selecting software enhancements that deliver value to users. The forecast model created in this study can be used to provide evidence of the impact on users of software enhancements. Further enhancements to the SVARIMA model created in this study can be made to not just measure the issues logged by users after releases but also the popularity of an application as releases provide bug remediations and features that impact the functionality of an application.

Chapter 6

Conclusion

6.1 Introduction

In this chapter, a summary of this thesis and insights gained are presented. The research questions posed in this study represent a real-world problem common to organisations in the software development industry, which is, delivering value that is important or relevant to software users. The problem represented by the research questions required an evidence-based approach which could use an organisations software development activities and measure its impact to users. This thesis used an evidence-based approach to identify relationships between software development activities and user feedback to produce a time series predictive model that can illustrate the impact of software changes on users.

6.2 Conclusion

The SVARIMA predictive model detailed in Chapter 3 is an artefact of this study which software development teams can use to select software updates that add value with confidence. Software development teams can use the forecast model by ingesting Git

events from software development teams source control and job ticketing systems to generate a model which shows the impact of their development activities on customers. Identifying enhancements released that result in lower issues and feature requests logged by users can help software development teams determine what value customers seek from their applications.

The model produced in this study shows that bad releases result in users logging bugs or features to enhance functionality that is sub-optimal. Using this model, software development teams can first use the historical data within the model to show the impact of previous releases on user satisfaction, experiment with prioritising software enhancement, release and capture feedback data to regenerate the model. Regenerated models can be compared to determine the impact of releases where forecasted issue numbers may be dropping compared to previous models as new types of features are selected for implementation that are relevant to users.

The forecast model produced in this study has real-world applicability as it solves a problem common to software development teams. Also, the data source used, Git, is commonly used in software development meaning the forecasting model has high interoperability with source control tools commonly used in the software development industry. This lowers the barrier to adoption by simplifying implementation. The outputs of this thesis have exceeded expectations as research questions have been answered and an artefact that has real-world applicability exists.

6.3 Challenges

The volume of data on GitHub produced a challenge in ingesting information into a database, transforming data into formats appropriate for statistical validation and modelling and processing data to create forecasting models. These challenges ranged from limitations of computing resources due to large volumes of data such as disk

storage or RAM through to the time taken to download and process data. This resulted in slower iterations of analysis occurring due to resource and time constraints.

The large volume of data also introduced challenges to exploratory analysis due to slow query times and data quality issues. The quality of data differed between GitHub projects, where some projects clearly categorised the types of issues logged where as others had different approaches to categorising issues due to large numbers of users logging issues differently. Due to time constraints, this study could not explore this observation in more detail however, this could be a future research area.

6.4 Future Research

A possible direction of future research is the trial and analysis of the forecasting model in the real world. Software delivery teams could be given insights from this model applied to their data, and changes in their operations could be made to measure the impact of the forecasting model on the software they produce and users of their applications. Business impact could also be measured to determine the financial or reputation changes to an organisation that uses evidence-based approaches to selecting software enhancements.

A limitation of this study was the difficulty in distinguishing between Bugs and Features due to data quality issues. To address this limitation, future research could develop tools that can classify GitHub event datasets. Tools developed could be used to support the automation of ingesting data into predictive models to generate near real-time insights into the impact of software development activities on teams and organisations, or what-if modelling further discussed in this section.

Another area of research could be a deeper examination of the types of Issues logged by users in GitHub and how the predictive model could be adjusted to carry out what-if modelling. The ability to test ideas on software enhancement selection against a forecast model to predict the results of these changes would allow software delivery teams to

quickly re-organise their backlogs to produce the most impactful releases to their users. This could result in reduced waste of software delivery resources and negative customer impact.

Enhancing the forecasting model produced in this study to ingest near real-time data and provide continual forecasts of the impact of code changes is a possible future area of research. Embedding the forecasting model into the software delivery process with near realtime insights could give software developers insight into the impact of their contributions to their customers. This could be used to improve software development practices and provide insights into training requirements for developers.

References

Aborass, D., Abu Hassan, H., Sahalash, I. & Al-Rimmawi, H. (2022). Application of arima models in forecasting average monthly rainfall in birzeit, palestine. *International Journal of Water Resources and Arid Environments*.

Alahyari, H., Svensson, R. B. & Gorschek, T. (2017). A study of value in agile software development organizations. *Journal of Systems and Software*, *125*, 271–288. Retrieved from https://doi.org/10.1016/j.jss.2016.12.007 doi: 10.1016/j.jss.2016.12.007

AlMarzouq, M., AlZaidan, A. & AlDallal, J. (2020). Mining github for research and education: challenges and opportunities. *International Journal of Web Information Systems*, *16*(4), 451–473.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... Thomas, D. (2001). *Manifesto for agile software development* [Internet web page]. Retrieved from https://agilemanifesto.org

Biffl, S., Aurum, A., Boehm, B., Erdogmus, H. & Grünbacher, P. (2006). *Value-based software engineering* (Vol. 1). Springer. Retrieved from https://doi.org/10.1007/3-540-29263-2_1 doi: 10.1007/3-540-29263-2_1

Binzagr, F., Labbaci, H. & Medjahed, B. (2019). Fame: An influencer model for service-oriented environments. In S. Yangui, I. B. Rodriguez, K. Drira & Z. Tari (Eds.), *Service-oriented computing* (pp. 216–230). Springer Cham. Retrieved from https://doi.org/10.1007/978-3-030-33702-5_3 doi: 10.1007/978-3-030-33702-5_3

Bissyande, T. F., Lo, D., Jiang, L., Réveillère, L., Klein, J. & Traon, Y. L. (2013, Nov). Got issues? who cares about it? a large scale investigation of issue trackers from github. In *2013 ieee 24th international symposium on software reliability engineering (issre)* (p. 188-197). doi: 10.1109/ISSRE.2013.6698918

Bucena, I. & Kirikowa, M. (2019). Simplifying the devops adoption process. In *Ceur workshop*. http://ceur-ws.org/Vol-1898/paper14.pdf.

Cryer, J. D. & Kellet, N. (1991). *Time series analysis*. Springer. doi: 10.1007/978-0-387-75959-3

Dabral, P. & Murry, M. Z. (2017). Modelling and forecasting of rainfall time series using sarima. *Environmental Processes*, *4*(2), 399–419.

Du, K., Yang, H., Zhang, Y., Duan, H., Wang, H., Hao, S., ... Yang, M. (2020). Understanding promotion-as-a-service on github. In *Annual computer security applications conference* (pp. 597–610). doi: 10.1145/3427228.3427258

Ebert, C. (2014). Software product management. *IEEE Software*, *31*(3), 21-24. doi: 10.1109/MS.2014.72

Farrugia, P., Petrisor, B., Farrokhyar, F. & Bhandari, M. (2010). Research questions, hypotheses and objectives. *Canadian journal of surgery*, *53*(4), 278.

Ferrante, D. (2006). Software licensing models: What's out there. *IT Professional*, *8*(6), 24-29. doi: 10.1109/MITP.2006.147

Fitrianti, H., Belwawin, S., Riyana, M. & Amin, R. (2019). Climate modeling using vector moving average autoregressive. In *Iop conference series: Earth and environmental science* (Vol. 343, p. 012201).

Fricker, S. A. (2012). Software product management. In *Software for people* (pp. 53–81). Springer. Retrieved from https://doi-org.ezproxy.aut.ac.nz/10.1007/978-3-642-55140-6 doi: 10.1007/978-3-642-55140-6

GitHub. (2022a, 7 July). *About issues* [Internet web page]. Author. Retrieved from https://docs.github.com/en/issues/tracking-your-work-with-issues/about-issues

GitHub. (2022b, 7 July). *Saving repositories with stars* [Internet web page]. Author. Retrieved from https://docs.github.com/en/get-started/exploring-projects-on-github/saving-repositories-with-stars

Gousios, G. & Spinellis, D. (2017, May). Mining software engineering data from github. In *2017 ieee/acm 39th international conference on software engineering companion (icse-c)* (p. 501-502). doi: 10.1109/ICSE-C.2017.164

Hagiwara, M. & Mita, M. (2019). *Github typo corpus: A large-scale multilingual dataset of misspellings and grammatical errors.*

Hahmann, M. (2019). Big data analysis techniques. In S. Sakr & A. Y. Zomaya (Eds.), *Encyclopedia of big data technologies* (pp. 180–184). Cham: Springer International Publishing. Retrieved from https://doi.org/10.1007/978-3-319-77525-8_279 doi: 10.1007/978-3-319-77525-8_279

Hair, J. F., Black, W. C., Babin, B. J. & Anderson, R. E. (2013). *Multivariate data analysis*. Pearson Education Limited. Retrieved from https://books.google.co.nz/books?id=VvXZnQEACAAJ

Hsi, I. & Potts, C. (2000). Studying the evolution and enhancement of software features. In *Proceedings 2000 international conference on software maintenance* (p. 143-151). doi: 10.1109/ICSM.2000.883033

IEEE. (1983). Ieee standard glossary of software engineering terminology. *ANSI/ IEEE Std 729-1983*, 1-40. doi: 10.1109/IEEESTD.1983.7435207

Jarczyk, O., Gruszka, B., Jaroszewicz, S., Bukowski, L. & Wierzbicki, A. (2014). Github projects. quality analysis of open-source software. In *Social informatics: 6th international conference, socinfo 2014, barcelona, spain, november 11-13, 2014. proceedings 6* (pp. 80–94).

Joca. (2016). *What is a software product* [Internet web page]. Gyaco. Retrieved from https://www.gyaco.com/2016/04/what-is-a-software-product

Kalliamvakou, E., Gousios, G., Spinellis, D. & Nancy, P. (2009).

Measuring developer contribution from software development repository. In *Mediterranean conference on information systems.* Retrieved from https://aisel.aisnet.org/cgi/viewcontent.cgi?referer=&httpsredir=1&article=1053&context=mcis2009

Kothari, C. R. (2004). *Research methodology: Methods and techniques*. https://ebookcentral.proquest.com/lib/aut/reader.action?docID=431524: New Age International.

Krishnan, M. S. (1993). Cost, quality and user satisfaction of software products: An empirical analysis. In *Proceedings of the 1993 conference of the centre for advanced studies on collaborative research: Software engineering - volume 1* (p. 400–411). IBM Press.

Kumar Dubey, A., Kumar, A., García-Díaz, V., Kumar Sharma, A. & Kanhaiya, K. (2021). Study and analysis of sarima and lstm in forecasting time series data. *Sustainable Energy Technologies and Assessments*, *47*, 101474. Retrieved from https://www.sciencedirect.com/science/article/pii/S2213138821004847 doi: https://doi.org/10.1016/j.seta.2021.101474

Li, Q., Zhao, L., Xue, Y. & Feng, L. (2021). Stress-buffering pattern of positive events on adolescents: An exploratory study based on social networks. *Computers in Human Behavior*, *114*, 106565. Retrieved from https://www.sciencedirect.com/science/article/pii/S0747563220303137 doi: https://doi.org/10.1016/j.chb.2020.106565

Li, Z. S., Arony, N. N., Devathasan, K., Sihag, M., Ernst, N. & Damian, D. (2024). Unveiling the life cycle of user feedback: Best practices from software practitioners. In *Proceedings of the 46th ieee/acm international conference on software engineering* (pp. 1–13). doi: 10.1145/3597503.3623309

Lin, C. & Huang, Z. (2009). A flexible metric-driven framework for software process. In *2009 fifth international joint conference on inc, ims and idc* (p. 1198-1202). doi: 10.1109/NCM.2009.189

Maalej, W., Nayebi, M. & Ruhe, G. (2019). Data-driven requirements engineering - an update. In *2019 ieee/acm 41st international conference on software engineering: Software engineering in practice (icse-seip)* (p. 289-290). doi: 10.1109/ICSE-SEIP.2019.00041

Mahdi, E., Provost, S. B., Salha, R. B. & Nashwan, I. I. (2017). Multivariate time series modeling of monthly rainfall amounts. *Electronic Journal of Applied Statistical Analysis*, *10*(1), 65–81.

Malki, A., Atlam, E.-S., Hassanien, A. E., Ewis, A., Dagnew, G. & Gad, I. (2022). Sarima model-based forecasting required number of covid-19 vaccines globally and empirical analysis of peoples' view towards the vaccines. *Alexandria Engineering Journal*, *61*(12), 12091-12110. Retrieved from https://www.sciencedirect.com/science/article/pii/S1110016822003714 doi: https://doi.org/10.1016/j.aej.2022.05.051

McCall, J. A., Richards, P. K. & Walters, G. F. (1977). *Factors in software quality. volume i. concepts and definitions of software quality* (Tech. Rep.). https://apps.dtic.mil/sti/pdfs/ADA049014.pdf: GENERAL ELECTRIC CO SUNNYVALE CA.

Microsoft. (2018, June). [Internet web page]. Author. Retrieved from https://news.microsoft.com/2018/06/04/microsoft-to-acquire-github-for-7-5-billion/

Miguel, J. P., Mauricio, D. & Rodríguez, G. (2014, nov). A review of software quality models for the evaluation of software products. *International Journal of Software Engineering & Applications*, *5*(6), 31–53. Retrieved from https://doi.org/10.5121%2Fijsea.2014.5603 doi: 10.5121/ijsea.2014.5603

Niedermaier, S., Koetter, F., Freymann, A. & Wagner, S. (2019). On observability and monitoring of distributed systems - an industry interview study. In S. Yangui, I. B. Rodriguez, K. Drira & Z. Tari (Eds.), *Service-oriented computing* (pp. 36–53). Springer Cham. Retrieved from https://doi.org/10.1007/978-3-030-33702-5_3 doi: 10.1007/978-3-030-33702-5_3

Olsson, H. H. & Bosch, J. (2014). From opinions to data-driven software r&d: A multi-case study on how to close the 'open loop' problem. In *2014 40th euromicro conference on software engineering and advanced applications* (p. 9-16). doi: 10.1109/SEAA.2014.75

Padhma, M. (2021, 12 July). [Internet web page]. Analytics Vidhya. Retrieved from https://www.analyticsvidhya.com/blog/2021/07/analyzing-popular-repositories-on-github

Peffers, K., Tuunanen, T., Rothenberger, M. A. & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of management information systems*, *24*(3), 45–77.

Poppendieck, M. (2007). Lean software development. In *Companion to the proceedings of the 29th international conference on software engineering* (p. 165–166). USA: IEEE Computer Society. Retrieved from https://doi.org/10.1109/ICSECOMPANION.2007.46 doi: 10.1109/ICSECOMPANION.2007.46

ProgrammableWeb. (n.d.). *What is an api* [Internet web page]. ProgrammableWeb: Author. Retrieved from https://www.programmableweb.com/category/all/apis

Ramzan, M., Jaffar, M. A. & Shahid, A. A. (2009). Value assignment process (vap): Establishing value of software through a new definition of value. In *Proceedings of the 4th international conference on ubiquitous information technologies & applications* (p. 1-8). Retrieved from https://doi-org.ezproxy.aut.ac.nz/10.1109/ICUT.2009.5405667 doi: 10.1109/ICUT.2009.5405667

Rawashdeh, A. & Matalkah, B. (2006). A new software quality model for evaluating cots components. *Journal of Computer Science*, *2*(4), 373–381.

Samoladas, I., Gousios, G., Spinellis, D. & Stamelos, I. (2008). The sqo-oss quality model: measurement based open source software evaluation. In *Ifip international conference on open source systems* (pp. 237–248). doi: 10.1007/978-0-387-09684-1_19

Shrivastava, S. K. & Shrivastava, C. (2024). Emerging software and tools in higher education institutions. *International Journal of Soft Computing and Engineering*, *13*(6), 1–6. doi: 10.35940/ijsce.F3620.13060124

Smith, D., Villalba, D., Irvine, M., Stanke, D. & Harvey, N. (2021). *State of devops 2021* [Status report]. Mountain View, San Francisco Bay Area. Retrieved from https://services.google.com/fh/files/misc/state-of-devops-2021.pdf

Sordi, J. O. D. (2021). Design science research method. In *Design science research methodology: Theory development from artifacts.* Springer International Publishing. Retrieved from https://doi.org/10.1007/978-3-030-82156-2_5 doi: 10.1007/978-3-030-82156-2_5

Sorjamaa, A., Hao, J., Reyhani, N., Ji, Y. & Lendasse, A. (2007). Methodology for long-term prediction of time series. *Neurocomputing*, *70*(16), 2861-2869. Retrieved from https://www.sciencedirect.com/science/article/pii/S0925231207001610 (Neural Network Applications in Electrical Engineering Selected papers from the 3rd International Work-Conference on Artificial Neural Networks (IWANN 2005)) doi: https://doi.org/10.1016/j.neucom.2006.06.015

Varuna, T. V. & Mohan, A. (2019). Trend prediction of github using time series analysis. In *2019 10th international conference on computing, communication and networking technologies (icccnt)* (p. 1-7). doi: 10.1109/ICCCNT45670.2019.8944878

Wang, J., Axtell, R. L. & Loerch, A. (2017). Developing winning marketing strategies in a competitive software market. In *Proceedings of the summer simulation multi-conference.* San Diego, CA, USA: Society for Computer Simulation International.

Wessel, M., Vargovich, J., Gerosa, M. A. & Treude, C. (2023). Github actions: the impact on the pull request process. *Empirical Software Engineering*, *28*(6), 131. doi: 10.1007/s10664-023-10369-w

Wieringa, R. J. (2014). *Design science methodology for information systems and software engineering*. Springer.

Wlodarski, R., Poniszewska-Maranda, A. & Falleri, J. (2022). Impact of software development processes on the outcome of student computing projects: A tale of two universities. *Information and Software Technology*, *144*(1). doi: 10.1016/j.infsof.2021.106787

Xu, L. & Brinkkemper, S. (2005). Concepts of product software: Paving the road for urgently needed research. In *The first international workshop on philosophical foundations of information systems engineering (phise'05)* (pp. 523–528). http://www.kybele.etsii.urjc.es/PHISE05/papers/sesionIII/XuBrinkkemper.pdf.

Yang, L. & Zhang, J. (2008). Research on project resource allocation models based on optimal software qualities. In *2008 4th international conference on wireless communications, networking and mobile computing* (p. 1-4). doi: 10.1109/WiCom.2008.1751

Zhang, T., Chen, J., Luo, X. & Li, T. (2019, Jan). Bug reports for desktop software and

mobile apps in github: What's the difference? *IEEE Software*, *36*(1), 63-71. doi: 10.1109/MS.2017.377142400