UTS UNIVERSITY OF TECHNOLOGY SYDNEY

Predicting Open Source Forked Pattern Survivability

by Bee Bee Chua

Thesis submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy

under the supervision of Professor Ying Zhang & Associate Professor Lu Qin

University of Technology Sydney
Faculty of Engineering and Information Technology

October 2021

CERTIFICATE OF ORIGINAL AUTHORSHIP

I, Bee Bee Chua, declare that this thesis, is submitted in fulfilment of the requirements for the award of Doctor of Philosophy in Information Technology, in the School of Computer Science at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

Signature: Production Note:
Signature removed prior to publication.

Date: 21st October 2021

Acknowledgements

Really, I have many people I wish to say thank you but to the very special and honourable ones are below:

Firstly, my research supervisory team, Professor Ying Zhang and Associate Professor Lu Qin, for their kind guidance and support throughout my PhD journey. Especially to Professor Zhang, a truly amazing and highly intellectual supervisor who I find to be extremely hardworking, critical, humble and knowledgeable. I am deeply indebted and wish to acknowledge his high-quality supervisory efforts, support, encouragement, belief and trust in me that I can produce good quality research.

Secondly, my family. I am grateful to my parents and my family members, including five adorable nieces and a nephew, for their continued love and support. To my special god daughter Maris Stella who I fondly miss.

To the team of medical specialists for their amazing expertise in healing my neck and right arm injury. My neurosurgeon Dr. Prakash Damodaran, my pain specialist Dr. Hasher Kadvil, my general practitioners Dr. Pramod Singh and Dr. Deep Kumar for a successful medical procedure and to my most respectful hand therapist Miss Tamara Carter for her hard work on healing my neck and hand. To all nurses and front desk receptionist staff for cheering me up during that most difficult and challenging time. Without them, my thesis writing is impossible.

To my research collaborators and mentors at UTS for advice and teaching collaboration, including Mr. Ravindra Bagia, Professor Roger Hadgraft, Dr. Danilo Valeros Bernardo, Dr. Jane Brennan. Dr. Laurel Dyson, Dr. Yulei Sui and Dr. Wentao. Li.

To many other researchers who I know sincerely to thank them for their esteemed support and encouragement: Professor June Verner, Professor Fethi Rabhi, Professor Paul Rowland, Professor Aileen Cartel-Steel, Associate Professor Shannon Kennedy-Clark, Professor Mehregan Mahdavi and Associate Professor Andrew Levula.

My heartfelt gratitude to Dr. Amy Nisselle for her expertise on guiding me how to write for an academic audience, copyediting and proofing my conference papers and thesis.

Finally, my deepest gratitude to Dr. Danilo Valeros Bernardo for his years of unwavering endearment and patience with me. Without his words of kind encouragement and motivation, I would not be able to achieve this degree.

List of Publications

This thesis comprised of a series of published and to-be-published articles together with an exegesis. The list of the publications is as follows:

1. B. B. Chua and Y. Zhang, “Applying a systematic literature review and content analysis method to analyse open source developers’ forking motivation interpretation, categories and consequences.” Journal of Australasian Information Systems, 2020.Vol. 24 No.1, pp. 1–19.

2. B. B. Chua, and Y. Zhang, “Predicting open source programming language repository file survivability from forking data.” OpenSym’19: Proceedings of the 15th International Symposium on Open Collaboration. 2019. Skövde Sweden

3. B. B. Chua, “A survey paper on open source forking motivation reasons and challenges.” Conference Proceedings of the Pacific Asia Conference of Information Systems (PACIS), 2017, Langkawi, Malaysia

4. B. B. Chua. Detecting sustainable programming languages through forking on open source projects for survivability. Proceedings of the IEEE International Symposium on Software Reliability Engineering (ISSRE) in conjunction with a WOSAR workshop, IEEE, 2015, Gaithersburg, USA. 120–124

5. B. B. Chua and Y. Zhang. “Healthy Fork File Repository (HFFR) Performance Prediction”. Journal of Systems and Information Technology (JST) Elsevier, Under review.

Other relevant publications developed but not included in this thesis include:

1. B. B. Chua. "Analysing Version Control Open-Source Software Survivability". Proceedings of the 19th International Conference on Distributed Multimedia Systems, DMS 2013, August 8-10 2013, Brighton, UK. Knowledge Systems Institute.
2. B. B. Chua and D.V. Bernardo. Open-Source Developer Download Tiers: A Survival Framework. 13th IEEE International Conference on IT Convergence and Security, ICITCS, 2013, Macau, China.

Contents

.... i
CERTIFICATE OF ORIGINAL AUTHORSHIP ii
Acknowledgements iv
List of Publications vi
Contents viii
List of Figures xi
List of Tables xii
List of Acronyms xiv
Abstract 1
Chapter 1 3
- 1.0 Introduction 3
- 1.1 Background 5
- 1.2 Research Motivation 6
- 1.3 Research Contributions 7

Chapter 2: Forking Literature Survey 8
- 2.1 Overview 8
- 2.2 Motivation 8
- 2.3 Approaches 8
- 2.4 Introduction 9
- 2.5 Research Study Motivation and Research Questions 12
  - 2.5.1 Research study motivation 12
  - 2.5.2 Research questions 13
- 2.6 Methodology: Systematic Literature Review and Content Analysis Method 15
  - 2.6.1 Systematic literature review search criteria 17
  - 2.6.2 Search strategy 18
  - 2.6.3 Methodological framework 20
  - 2.6.4 Content analysis method 21
- 2.7 Forking Motivation Interpretations 22
  - 2.7.1 How do researchers interpret developer forking and categorise forking motivational behaviour? 23
  - 2.7.2 What were the most popular methodologies used by forking researchers from 1990 to 2017? 32

2.7.3 What aspects of open source forking have been researched and reported? 32
2.7.4 Newcomers or new developers forking motivation from 2020 to 2021 34
2.7.5 Shifting motivation through time and journey 35
2.7.6 Shifting forking motivation 36
2.8 Summary from the literature survey 37
Chapter 3: Literature Survey Research Methodology 40
3.1 Overview 40
3.2 Motivation 40
3.3 Introduction 40
3.4 Literature Survey Selection Criteria and Categorisation 41
3.5 Category I: Survey-based Research Methodology 42
3.6 Category II: Data Mining Algorithm-based Research Methodology 50
3.7 Category III: Machine Learning Algorithm-based Research Methodology 58
3.8 Machine Learning: A K Nearest Neighbour Method 65
3.8.1 Euclidean distance metric 66
3.8.2 Adopting Euclidean distance: characteristics identification and rationale 66
3.8.3 Identifying Euclidean distance characteristics 67
3.8.4 Our research dataset characteristics 67
Chapter 4: Models 69
4.1 Overview 69
4.2 Literature Survey Road Map Model 69
4.3 Chua and Zhang Open Source Software Forking Pattern Prediction Model 70
Chapter 5: A Pilot Study 72
5.1 Overview 72
5.2 Motivation 72
5.3 Background 73
5.4 Forking Patterns 74
5.5 Software Survival and Programming Language Survival Importance 75
5.6 Survivability Prediction on the K Nearest Neighbour Method 77
5.7 Programming Language Repository File Categorisation and Fork Pattern Classifiers 81
5.8 Classifier Results 82
5.9 K Nearest Neighbour Results 83

5.9.1 Case One 84

5.9.2 Case Two 85

5.9.3 Case Three 85

5.9.4 Case Four 85

5.10 Evaluation 86

5.11 Conclusions and Future Work 88

Chapter 6: A Longitudinal Study 90

6.1 Overview 90

6. 2 Motivation 90

6.3 Background 91

6.4 Fork Pattern Identification and Data Collection 92

6.5 Normalisation and Euclidean Distance 95

6.6 Results 100

6.7 Evaluative Test Results 102

6.8 Discussion 104

Chapter 7: Conclusion 106

7.1 Overview 106

7.2 Contributions 107

7.3 Recommendations 108

7.4 Future Work 109

Bibliography 110

List of Figures

Figure 2. 1: Combined approaches: systematic literature review and content analysis method 16
Figure 2. 2: The systematic literature review search strategy for research papers........ 19
Figure 2. 3: Data collection methods in the 21 papers........ 32
Figure 2. 4: Units of analysis in the 21 papers........ 33
Figure 2. 5: Forking lessons learnt across the 21 papers........ 34
Figure 2. 6: The open source developers' motivation movement........ 36

Figure 3. 1: Paper selection criteria 42

Figure 4. 1: Literature survey mapping model........ 70
Figure 4. 2: The Chua and Zhang OSS forking pattern prediction model 71

Figure 5. 1: Categorising programming language repository file forks as short- or long-lived. 82
Figure 5. 2: Evaluative results comparison of the dataset........ 88

Figure 6. 1: Euclidean distance ranking........ 102
Figure 6. 2: Evaluative results 104

List of Tables

Table 2. 1: The systematic literature review identified 21 relevant and suitable papers 19

Table 2. 2 A forking motivation methodological framework 20

Table 2. 3: Forking interpretation types 21

Table 2. 4: Fork categorisation, sustainability and lessons learnt 28

Table 3. 1: Literature Survey Research Methodology in OSS 46

Table 3. 2: Data Mining algorithm-based type research methodology 55

Table 3. 3: Machine learning research-based methodology in OSS 61

Table 3. 4: Four widely-adopted KNN distance metrics... 66

Table 5. 1: Fork patterns 75

Table 5. 2: Variables defined for programming language survivability 79

Table 5. 3: Forking patterns 81

Table 5. 4: Categorising programming language repository files forks as short- or long-lived...82

Table 5.5: Categorising programming language repository files sorted by Euclidean distance 84

Table 5. 6: Environment compliance 86

Table 6. 1: Examples of file repository monthly forking.. 93

Table 6. 2: Big query statement 93

Table 6. 3: Forking data of selected file repositories, 2015–2020 94

Table 6. 4: Variables defined for a healthy fork file repository... 96

Table 6. 5: Forking in 5 years (2015-2020) after normalisation 97

Table 6. 6: Healthy fork file repository types and counts 99

Table 6. 7: Healthy fork file repository types ranked by Euclidean distance 100

Table 6. 8: Healthy fork file repository types ranked by Euclidean distance 101

Table 6. 9: Definition and formula for accuracy, precision, sensitivity and specificity 103

List of Acronyms

CAM	Content Analysis Method
CVS	Control Version System
FN	False Negative
FP	False Positive
HFFR	Healthy File Fork Repository
KNN	K Nearest Neighbour Method
OS	Open Source
OSS	Open Source Software
SLR	Systematic Literature Review
SPF	Specific Repository File
SRFHF	Specific Repository File that did not meet the full environment licence but has Healthy Fork
SRFMSPL	Specific Repository File that met official licence compliance and adopted a Modern Sustainable Programming Language
SRFOL	Specific Repository File that met Official Licence compliance
SRFOLHF	Specific Repository File that met Official Licence compliance that has Healthy Fork
SRFOLMSPLHF	Specific Repository File that met Official Licence compliance and adopted a Modern Sustainable Programming Language that has Healthy Fork
SRFOLTSPL	Specific Repository File that met Official Licence and adopted a Traditional Sustainable Programming Language
SRFTSPL	Specific Repository File that adopted a Traditional Sustainable Programming Language
SRFTSPLHF	Specific Repository File that adopted a Traditional Sustainable Programming Language that has Healthy Fork
TN	True Negative
TP	True Positive
VT	Virus Total

Abstract

The motivational behaviour of open source (OS) developers has always been an active focus of research. With the introduction of the forking technique a related research area of developer forking motivational behaviour has gained significance, partly due to the problem of forking scarcity and low fork visibility performance.

The objective of forking is to improve and innovate source code quality from voluntary developers. Unfortunately, the forking technique is not very sustainable in improving fork efficiency and efficacy. Further, developers may spend time forking source codes that may become inactive and consequently prove to be a waste of time and effort. From the perspective of project owners, if their repositories do not receive a good fork response from developers, their repositories will not grow.

This doctoral research study aimed to address these problems by avoiding forking scarcity, increasing high fork visibility performance, and promoting positive developer forking motivation. We also needed to investigate OS environment compliance to determine whether it contributes to improved fork visibility, reduced fork deficiency and/or is viewed positively by developers.

The research approach was to apply a model to predict high fork visibility. The model is based on the K Nearest Neighbour machine learning algorithm, using the Euclidean distance metric to predict high fork visibility performance. We piloted it using nine repository classifiers and then conducted a longitudinal study of five select repository classifiers to determine accuracy and distance approximation. Our work adds a new body of knowledge to OS forking theory and provides a deeper understanding of developer forking motivational behaviour.

In the first phase of this study, we conducted a literature review of forking motivation and research methods used in OSS. We then developed and tested our model. In the last phase, we identified OSS patterns and detected fork longevity to determine whether environmental compliance was fully, partially or not at all satisfied. Most importantly, we showed that high fork visibility environmental compliance distance approximation can positively predict developer forking interest.

Chapter 1

1.0 Introduction

This chapter introduces the research area, background to the research questions, the motivation to conduct the research, and contributions to the discipline.

Forking scarcity and low fork visibility are two serious problems for developers aiming to produce high-quality, open source (OS) software (OSS). Forking is a useful technique for developers that encourages them to contribute to modifying or fixing codes or making recommendations to enhance original project file repositories [1]. However, when forking is not used effectively it can lead to substantial code development wastage; additionally, low fork visibility for repositories can cause resource allocation issues that can impact repository survivability. Investigating developers' forking motivation and behaviour can help reduce forking scarcity and low repository forking visibility. Further, developing a predictive forking technique can promote forking efficiency and effectiveness.

A number of reputable OS hosting platforms such as GitHub [2], SourceForge [3] and Bitbucket [4] offer project owners create a file repository by allowing them to host their source files. These platforms also aim to attract developers to voluntarily fork and contribute to software development.

In the existing forking literature, many studies [5-8] indicate that developer forking interest or motivation is important. To comprehend factors that may influence developer interest or motivation or interest, we reviewed the OS literature to identify and categorise developers' forking behaviour. Self-development factors related to individual developer learnability and the domain knowledge of a project, as well as variables relating to programming language, discussed widely in the literature over two decades.

Other studies [8-12] identified additional personal variables such as developers seeking a coding career, to network with other experienced developers, or to form a new coding group. Discontinuing to fork a file repository was associated with involvement in a conflicting OS project or team or leadership dispute, or commercial, legal or political reasons [6, 13-15].

There are limited studies [16-18] on how OS licence restriction, less sustainable programming languages or less innovative frontiers technology may affect developer forking behaviour. There are no studies that aim to understand the impact of OS forking environmental compliance factors on developer forking behaviour. For instance, the top three important OS variables on quality programming languages, OS licence compliance and the new or emerging technology trend [19-21] often recognised as prevalent topics discussed widely by communities. although the introduction of social networks is a recent and hot topic on understanding communities social interaction influences the way how developers fork but it was not discussed in literature heavily over the past two decades Moreover, research to date on predicting fork success outcomes were based in single variables rather than investigating how multiple variables influence outcomes.

The novelty of our research is predicting the accuracy of high fork visibility performance from repository populations that satisfy full OS environmental compliance, in response to developers' positive forking motivation and behaviour. Our research study encompasses forking features that are highly desirable OS environmental variables to analyse monthly forking data generated from two datasets downloaded from the GitHub database. We applied a supervised machine learning algorithm – the K Nearest Neighbour method – with the Euclidean Distance metric to predict the closest distance of high fork visibility performance on repositories with and without complete OS environmental compliance.

Our research results are original and novel and include several peer-reviewed publications. They are convincing and promising, using longitudinal data on monthly fork movement in repositories across a six-year period.

We are the first research group to contribute novel findings to the OS forking body of knowledge by applying a machine learning algorithm to predict healthy fork performance by analysing forking count to quantify positive developer motivation.

1.1 Background

Open source software development is a new technology platform that simplifies the process of collaborating source code writing between disparate developers. Its simplicity lies in a project owner who owns a piece of source code that can be uploaded and made visible in a hosting platform for other developers to fork, share and contribute. The OS process can not only help by reducing development time and costs but it also promotes the quality of source code through iterative improvement and innovation.

A number of reputable OS hosting platforms – such as such as GitHub [2], SourceForge [3] and Bitbucket [4] – offer project owners the opportunity to create file repositories, allowing them to host source files. These platforms also aim to attract developers to join, fork and contribute voluntarily. Different hosting platforms have different OS language, an OS licence and developers.

For example, GitHub infrastructure includes a declaration of programming language, OS licence and being a sizeable developer. Despite basic OS infrastructure settings being provided to project owners, the forking status of many file repositories is still not healthy. This could be due to insufficient developers or forking deficiency.

There are no associations reported in the literature between developer forking motivation and developer insufficiency or forking deficiency [22]. Personal reasons, commercial reasons and OS infrastructure settings are barriers that can impact on developer forking motivation [5, 7, 9-11]. Personal and commercial reasons may be subjective, as they may be preconceived, opinionated, discriminatory or prejudiced; hence these findings are less convincing in predicting the forking behaviour or larger populations of developers. OS infrastructure parameters are standardised, objective, unbiased and non-discriminatory, making them more practical and convincing to predict developer forking motivation on a large population.

OS infrastructure parameters may restrict or prohibit developers and reduce their motivation to fork. For instance, the OS licence permission restriction can hinder a developer forking an original repository into his or her own environment; a specific programming language used by a project owner may be unfamiliar to developers, so can reduce the chance of forking; or a new technology may be challenging in its complexity.

1.2 Research Motivation

This study was driven by three primary motivations. Firstly, to improve clarity on interpretation of developer forking behaviour. Clarification is required because different researchers interpret the term differently, and some do not have sufficient empirical data to support these varied interpretations. As such, we want to contribute a new body of knowledge from the theoretical understanding of forking, based on an OS environmental compliance perspective, which places importance on quantifying fork count as an indicator of developers' positive forking motivation.

The second motivation is to address the forking scarcity and low fork visibility performance for some repositories, so that they can survive.

The third is to develop a new predictive model based on programming language repository classifiers to detect high fork visibility performance.

1.3 Research Contributions

The three main contributions from our work are:

1. Improving the OS theory on the fundamental concept of developer forking motivation and behaviour interpretation by understanding how high fork visibility performance can positively predict programming language repository compliance from a classifier's perspective.
2. Introducing a new predictive forking model based on a machine learning approach that incorporates OS environmental compliance variables to predict high forking visibility performance as a way to judge developers' forking behaviour.
3. Solving the forking scarcity and low fork visibility problem by acknowledging three types of developer forking behaviour patterns: 1) fork once only; 2) fork intermittently; and 3) fork steadily.

Chapter 2: Forking Literature Survey

2.1 Overview

This chapter reviews the relevant literature, firstly relating to interpretations of OS developers' forking motivations, reasons and challenges. We then applied systematic literature review (SLR) [23, 24] and content analysis method (CAM) [25] methodological frameworks to investigate OS forking divergence to evaluate OSS developer forking motivation, how motivations are interpreted and, categorised, and consequences. This work was published as two research articles [26, 27].

2.2 Motivation

The primary motivation in conducting the literature survey was to critically examine the many interpretations of developer forking motivation and see which had a specific interpretation on evaluating forking to understand OS environmental parameters. We also aimed to contribute to the OSS forking community through publishing the results of the combined SLR and Content Analysis reviews to evaluate OS forking motivation reasons and challenges as there was no peer-reviewed research on this topic, despite several surveyed papers discussing motivational forking reasons from OS developers to contribute. We also sought to determine whether forking motivations differed for first-time developers versus others.

2.3 Approaches

Our literature review consisted of surveying OS forking paper published in two time periods: from 1990 to 2017, then from 2018 to 2021. The latter focused on understanding first-time developers' forking motivation.

In the first part of our literature survey, we adopted the SLR method [23, 24] to provide a rigorous and vigorous literature review, as the method can synthesise controversial views and dilemmas when discussing different perspectives on the same topic. SLR is one of the most reliable methods for conducting a software engineering literature review and is widely used in computer science, software engineering, social science and information systems research [28-30]. Software engineering researchers [23, 31] even proclaimed that SLR is a form of evidence-based software engineering that can address many engineering questions posed by researchers. Here we outline the process for conducting a SLR by specifying research questions, describing the search and retrieval process, collecting evidence, synthesising the evidence and providing results.

2.4 Introduction

GitHub is a hosting website for developing OSS through social coding by multiple developers. GitHub stores projects, files, programming languages, licences and developer profiles. In May 2014 GitHub was the largest hosting coding platform, reported as having over 37 million population users and over 100 million public and private repositories [32]. In 2017, GitHub had 26 million registered developers from 110,000 organisations and an additional 20 million developers and users visit GitHub daily without registering [33]. GitHub has long-term viability and remains on the cutting edge of technology, particularly the forking feature, which many developers adopt and use.

Forking is an important feature in GitHub, allowing developers to make a copy of original source code, download it into their own environment to learn from or make changes, then submit adapted code back to the project owners (sometimes referred to as 'upstream'). When a file is forked by developers in GitHub, the developer may indirectly adapt it to enhance the programming language longevity. Developers may download a programming

language not only because the language file repository is interesting and unique but because it also may have strong compliance and interoperability with local developmental environments.

However, most OS projects do not receive high forking counts and there is currently no reliable method of determining whether developer motivation behind projects with the most forked files is 'genuine' or 'non-genuine'. Genuine motivation would be developers who are willing to contribute, rewrite source codes and submit them upstream for owners to accept and merge; non-genuine developers would simply retain the code – adapted or not – for their own purposes, without submitting it upstream. Moreover, programming language use, adoption and forking varies, based on the number of projects and file repositories, so the evidence base on developer forking motivation behaviour is unclear.

A project can have one or multiple programming languages to allow one or more developers to create single or multiple file repositories. GitHub hosts 339 active programming languages yet less than one twelfth are sustainable or widely adopted in projects by organisations [1]. However, there are other factors beyond popular use that influence sustainability of a programming language, including organisational and project boundaries, the programming languages themselves, and above all, social psychology aspects such as developer motivation, preference, and interest. Flexible coding provides many software developments companies and developers the freedom to submit their source codes on GitHub and allow other developers to respond and fork the code.

Despite a number of published OS forking studies that highlight critical factors attributed to successful software forking and forking failure [7, 8, 13, 15, 34], there has been no systematic study mapping understanding of forking motivation, interpretation, categorisation and consequences. This paper therefore presents a systematic review of studies to compare, contrast, summarise and synthesise existing studies to inform future

decisions about OS forking research by providing an understanding of why some projects are forked more than others, through the lens of project and programming language characteristics.

There are currently few studies that have identified or classified developer forking motivation to enhance forking visibility, and little knowledge about potential differences in forking motivation between junior and senior developers across software engineering, computing science and information systems literature. Therefore, clarifications are required. There is no framework to categorise forking motivation behaviour and its effect on forking visibility. A methodological framework would be useful for researchers to implement sustainable ways to motivate developers to fork more programming language files.

The research objective was therefore to identify types of developer forking motivation and forking consequences cited in the existing OS literature through SLR adopted from [23] of conference papers and literature in relevant databases. A SLR uses specific search criteria to identify appropriate papers that are then read and analysed carefully using content analysis (a qualitative research technique) [35] to extract themes and words, in this instance, describing forking. Each paper is scrutinised to understand research methodology, methods of data collection, units of analysis and conclusions.

The contributions of this research include: 1) summarising the existing evidence base on forking motivation and consequences into a methodological framework; 2) providing guidelines for those interested in conducting research on understanding developer forking motivation and consequences influencing the ability of projects and organisations to predict project survivability and sustainability [survivability as in the duration of a programming language and sustainability as in measuring a programming language's continued use by developers]; 3) filling a gap on forking risk literature to inform future

research; and 4) proposing a strategy to map how forking motivation and programming language influence forking visibility. We aim to support OSS communities and researchers with theoretical insights on developer forking motivation, consequences and impacts.

2.5 Research Study Motivation and Research Questions

2.5.1 Research study motivation

This study was designed primarily to contribute to a theoretical understanding of OS forking and to potentially identify new influencing factors. It is important to address the current disparity in the literature around a theoretical understanding of what forking features and functions can offer in OSS, that is, perspectives on interpreting and defining forking as software, project, file repository and programming language source code. There is also a need to understand what influencing factors can cause OS project forking to succeed or fail. Forking activity has been reported using a variety of measures, including activity growth, developer interest and licensing [5, 7, 34, 36] but there are few analyses measuring forking motivation implicitly or explicitly. Moreover, there is limited evidence to confirm forking activeness in spin-off projects that may be strongly influenced by project topic, organisation and licence, or developer forking motivation (genuine or non-genuine). Further, a myriad of programming languages have tried to spur developer interest but not all succeed or sustain developer forking interest. Lastly, there is little evidence on whether genuine developers are more positively motivated to fork compared with non-genuine developers; for example, Murgia et al. [37] noted that developers have expressed love and joy when they fix OSS artefacts successfully, while other developers expressed anger, surprise, sadness or fear over challenging OSS artifacts.

2.5.2 Research questions

According to Jiang et al [7], they defined forking is copying a repository to create a new software repository. Software forking is increasingly adopted by many OSS communities for various reasons, including social and political. For instance, a relational database management system project – MYSQL, owned by Sun Microsystems – was forked into another project –, called Maria DB – due to uncertainty whether Oracle stewardship could maintain MYSQL's survivability [38].

For new OS projects, it is critical to seek developers' participation and collaboration. Interestingly, most junior developers prefer to fork new projects more than old projects, despite less involvement from senior developers, and junior developers seem to prioritise forking in favour of using new programming languages [1].

The number of terminated projects is also increasing due to low sustainable community participation and collaboration to fix bugs and improve features [7]. It is therefore important to identify types of developer forking motivational behaviour and risk to prevent project termination due to low developer interest. Identifying forking motivation may help communities increase sustainability and build more long-term contributors.

Three research questions (RQs) guided this study.

RQ1: How do researchers interpret forking and categorise developer forking motivational behaviour?

Types of developer motivation to fork OSS were captured to address RQ1, referencing a definition of 'motivational behaviour' as a reason or reasons for acting or behaving in a particular way [39]. As the topic is closely related to the study of human behaviour, databases spanning a variety of disciplines – such as humanities and social science,

management science, policy, psychology and sociology – were selected to search for OSS papers.

RQ2: What were the most popular methodologies used to research forking from 1990 to 2017?

The Open Source Software Initiative (OSI) [20] started in 1990 with support from many of the world's largest OSS projects and contributors. They are Mozilla Foundation, Free BSD Foundation, Debian, Drupal Association, Linux Foundation, Wikimedia Foundation and WordPress Foundation. While the evolution of forking started in 1990, it is unclear what forking research papers have been published over the past nearly three decades. Through RQ2 we therefore aim to provide up- to-date information on forking throughout the period of OS development.

RQ3: What aspects of OS forking have been researched and reported?

Open source forking is not a new topic but has gained popularity in recent years, with many researchers and communities interested in investigating forking reliability [7, 34]. When GitHub launched there was an overwhelming response from researchers investigating forking technique performance to analyse forking in sustainable projects by programming language committees or version control files [40]. Unfortunately, research findings remain unclear, particularly a lack of data to understand possible impacts and consequences of negative forking. Therefore, RQ3 sought to find barriers to forking to better guide further research.

2.6 Methodology: Systematic Literature Review and Content Analysis Method

The SLR method was employed to examine and review developers' motivational forking behaviour in OS literature as the topic has been published across multiple disciplines for a number of years. SLR was chosen to provide a rigorous and vigorous literature review, as the method can synthesise controversial views and dilemmas when discussing different perspectives on the same topic. SLR is one of the most reliable methods for conducting a software engineering literature review and is widely used in computer science, software engineering, social science and information systems research [28, 29]. Software engineering researchers [23, 24, 31] even proclaimed that SLR is a form of evidence-based software engineering that can address many engineering questions posed by researchers.

A SLR has several features. First, a research question is being addressed and a systematic method applied to perform the review. Second, a search strategy is defined to detect the relevance of retrieved literature as far as possible. Third, a search strategy is used to review documents to assess rigour, completeness and repeatability. Lastly, explicit and implicit criteria are listed before conducting a SLR. Here we outline the process we used when conducting a SLR by specifying research questions, describing the search and retrieval process, collecting evidence, synthesising the evidence and providing results.

Applying SLR guidelines provided discrete steps to locate and review appropriate documents describing OS forking motivation. As the content of each paper was comprehensive the Content Analysis method (CAM) was then applied to analyse and interpret articles (**Error! Reference source not found.**), as it is a reliable method for

analysing text data, themes or concepts, including intuitive, impressionistic and interpretive, quantifying them into systematic and strict textual analyses [25, 41].

Highly cited content analysis researchers [35] defined three approaches: 1) a conventional analysis based on text data, categorised into coding types; 2) a directed approach based on a theory or research findings, where user analysis begins with guidance for initial codes; and 3) a summative content analysis based on words or phrase count, compared by the underlying context interpretation.

Here we adopted a summative content analysis of the SLR articles to identify and count common themes and words used to describe forking motivation and sustainability (**Error! Reference source not found.**).

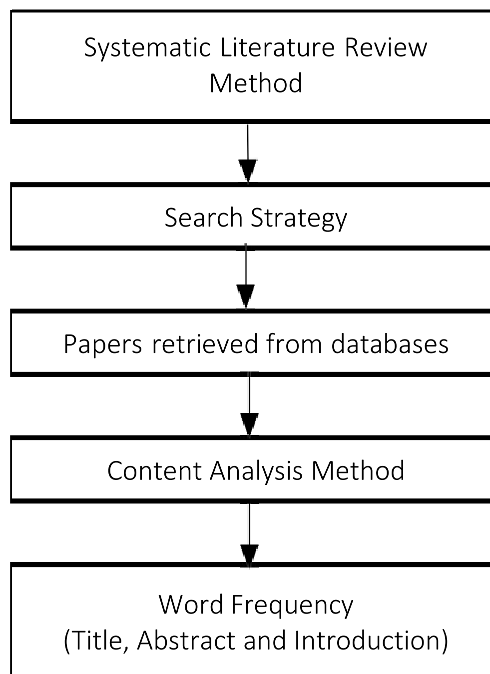

Figure 2. 1: Combined approaches: systematic literature review and content analysis method

2.6.1 Systematic literature review search criteria

To ensure the literature search was specific and to identify the most relevant, high-quality articles, in line with SLR guidelines [28, 29], the explicit and implicit criteria used to evaluate each study were:

- Peer-reviewed conference or journal papers, published and indexed either in Google Scholar, ACM, IEEE, Science Direct, Springer or MISQ; AND
- Written in English; AND
- Titles or content included phrases "open source forking motivation", "open source software forking", "open source project forking", "open source social forking", "open source code forking", "open source language forking" OR "file repository forking"; AND
- Published from 1990 to 2017; AND
- Published from top quality Information Systems Conferences or Journals; AND
- Described the research methodology used – systematic study, stratified sampling, case study, survey, interview, experiment, quasi-experiment or other study types – to collect, analyse and interpret results to address research questions in the paper. This criterion was necessary to determine common and similar research methodologies used by OSS researchers to inform the methods and reduce bias of method selection to study forking patterns, frequency, etc.

When searching for quality papers, exclusion criteria were articled that:

1. Were too short (e.g., less than five pages), general, based on a different perspective or did not include empirical evidence to demonstrate the authors' claim; OR

2. Did not identify positive and/or negative impacts or consequences of motivating factors, and did not discuss challenges or barriers, as the objective was to understand developer forking motivation. For example, if there was no discussion on positive or negative factors impacting developer forking motivation; OR
3. Did not include empirical evidence from the stated methods, be they quantitative, qualitative or mixed.

2.6.2 Search strategy

Two approaches were applied to conduct the SLR search (**Error! Reference source not found.**). The first search was conducted on 1 October 2017 on Google Scholar for the term "open source forking behaviour", resulting in 21,200 URLs. Because the first approach based on text searching is broad, the second search approach aimed to narrow the search on ACM, IEEE, SCOPUS and other databases, which have more OSS publications. Results were then sorted by relevance and filtered for papers published from 1996 to 2017, resulting in 9530 URLs. These papers were both peer-reviewed and non-peer-reviewed by multiple disciplines ranging from economics, management, information systems, software engineering and sociology [28-30].

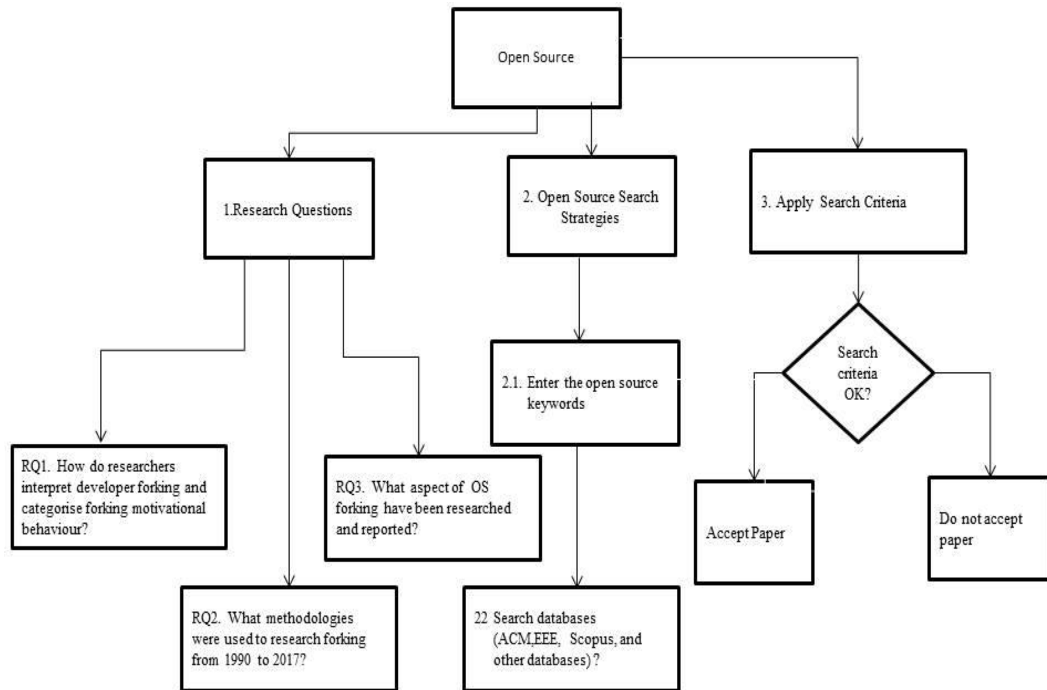

Figure 2. 2: *The systematic literature review search strategy for research papers*

As each Google Scholar results page lists 10 URLs linking to peer-reviewed articles cited in databases, the first five pages were reviewed by clicking each link to each URL, and the summary or abstract and introduction were read to confirm relevancy and suitability. In total, 13 papers were identified in ACM, IEEE, Science Direct and or MISQ databases plus 8 other relevant papers in other databases (**Error! Reference source not found.**).

Table 2. 1: The systematic literature review identified 21 relevant and suitable papers

Database	Number	Authors
Google Scholar	8	Biazzini & Baudry, 2014 [42]; Ernst et al., 2010 [40]; Fujita & Ikuine, 2014 [43]; Fung et al., 2012 [34]; Gamalielesson & Lundell, 2013 [8]; Ikuine & Fujita, 2014 [6]; Moen, 1999 [18]; Nyman et al., 2012 [44]
ACM	6	Dabbish et al., 2012 [36]; Glass, 2003 [15]; Neville-Neil, 2011 [45]; Nyman, 2014 [46]; Ray & Kim, 2012 [47]; Ray et al. 2014 [17]
IEEE	2	Chua, 2015 [48]; Cosentino et al., 2017 [14]
MISQ	1	Van Krogh et al., 2012 [9]
Springer	4	Azarbakht & Jensen, 2017 [13]; Jiang et al., 2017 [7]; Robles & Gonzalez-Barahona, 2012 [5]; Nyman & Mikkonen, 2011 [3]

2.6.3 Methodological framework

Of the 21 papers, five focused on forking sustainability, three on forking challenges and 17 on lessons learnt. Forking motivation, sustainability and lessons learnt were synthesised into a methodological framework with three steps to address the research questions via retrieval, categorisation and reporting (**Error! Reference source not found.**). 1) Identify variables used to define motivation and its interpretation from both broad and specific perspectives by applying the three RQs via the SLR to select and review papers. 2) Categorise forking interpretations into three categories (OS forking motivation, sustainability and lessons learnt) by applying the CAM using the same theme or word. 3) Group similar keywords and papers that describe the three categories of forking motivation, sustainability and lessons learnt. Conclusions were then drawn from these findings regarding forking challenges and lessons to be learnt.

Table 2. 2 A forking motivation methodological framework

Purpose	Process	Outcome
1. Identify variables that describe forking motivation and its interpretation	Apply SLR to select relevant papers from selective databases	**Retrieve** relevant papers on forking motivation
2. Categorise forking into motivation, sustainability and lessons learnt	Apply CAM and classify common themes or words	**Categorise** forking motivation into three classes
3. Group similar keywords to describe OS developer forking motivation, sustainability and forking lessons learnt	Analyse word count frequency (title, abstract and introduction)	**Report** forking motivation factors

2.6.4 Content analysis method

Each of the 21 papers identified was scrutinised for context using content analysis. Papers were first scanned to confirm the word 'fork*' was mentioned and the research evidence was empirical, then themes and key words were extracted. Next, each title was checked, abstract read, and adjectives that described 'fork*' quantified (**Error! Reference source not found.**). For example, when reviewing the papers "Code Forking in Open-Source Software: A Requirements Perspective" [40] and "Perspective on Code Forking and Sustainability in Open Source Software" [44] the word 'code' occurred twice so '2' was entered under 'code' forking type identified by the Google Scholar search in **Error! Reference source not found.**. Occurrences of forking motivation (n=10), forking sustainability (n=4), consequences (n=2), impacts (n=2) and threats (n=1) were also noted. Paper content was then analysed, noting research method, unit of analysis and results, then the introduction and conclusion were reviewed in more detail.

Table 2. 3: Forking interpretation types

	Paper identified via					
Forking type	**ACM**	**IEEE**	**Springer**	**MISQ**	**Google Scholar**	**TOTAL**
Open source				1		1
Project	4	1	1		1	7
Software		1			2	3
Social	2		1		2	5
Code			1		2	3
Language					1	1
File repository			1			1
TOTAL	6	2	4	1	8	21

Next, papers were grouped into four categories to address RQ1:

- *Developer forking interpretations*: 7 interpretations of forking (**Error! Reference source not found.**).
- *Developer motivation and reasons*: a subset of papers reported similar variables (Table 3). For instance, [7, 9, 15, 34] reported divergent specialisation, objective misalignment, poor governance and leadership and culture.
- *Forking sustainability*: four groups of researchers [3, 7, 8, 40] undertook real-world projects, comparing original versus forked projects (**Error! Reference source not found.**). Successful and sustainable projects included community-level projects, such as MariaDB forked by MYSQL, the software level of MS Word and LibreOffice and ecosystem levels of LibreOffice forked from OpenOffice.
- *Forking lessons learnt on project compatibility issues*: 19 papers cited forking lessons and seven described more than one type of forking reason, including no guidance or direction, copyright, licensing conflict, project ownership or dividing the forking community [6, 13-15, 18, 43, 45] pointed out that technical developers' roles are becoming specialised.

2.7 Forking Motivation Interpretations

Although a number of motivating factors identified in previous OS studies are applicable in the forking context, a number of diverse forking motivation factors were detected in this literature review, including project revival and alignment, culture traits, divergent specialisation, individual ownership, licence and software compliance, community disintegration, community practice and extending community social coding development. Therefore prior to investigating forking motivation factors, an additional research question was posed.

2.7.1 How do researchers interpret developer forking and categorise forking motivational behaviour?

These findings reveal a diversity of forking interpretations (**Error! Reference source not found.**), with project forking most common (7 papers), and OS, programming language and file repository the least (1 each). However, fork type was interpreted differently by different researchers, due to the metadata of the dataset they downloaded from the hosting server. For example, GitHub was the only hosting server to categorise file repository forking. To further understand the forking interpretation each paper, the categories were defined in more detail (paper classifications shown in **Error! Reference source not found.**).

2.7.1.1 Open Source Forking

The early 1990s saw a proliferation of research on OS motivation. Krogh et al. [9] reviewed seven years of publications and identified 40 papers that focused on OS developer motivation, including [49-52]. They synthesised findings across these papers into three classes of motivation: intrinsic, internalised intrinsic and extrinsic. Intrinsic motivation included ideology, altruism, kinship and fun, and can drive developers to fork software. Internalised intrinsic motivation included reputation, reciprocity, learning and own-use [49-52]. Extrinsic motivation may include being paid for the work or finding a career in coding [49-52]. Hippel and Von Krogh [53] and Goode [54, 55] studied organisational information sharing of OSS and innovation models as influencing factors on motivation between adopters and non-adopters. They found more reputable organisations and innovative projects are more likely to attract OSS developer attention to download or copy repository files.

2.7.1.2 Concept of Forking

Forking can be defined in different ways. Nyman et al. [3, 44, 46] defined it in a project context, where the development of an independent project is based on a developer who copies original source code from a software package. Ikuine and Fujita [6] defined software forking as the continuous development of software. Fung, Aurum and Tang [34] defined social forking to identify relationships within communities, and studied how forks are used to facilitate OSS development. Code forking is defined as a forked project copied from an existing code base and moved away from the original project leadership direction [3].

In our paper, we define language forking as a repository language that is copied by other developers. Defining programming language success varies from researcher to researcher. For instance, programming language interoperability performance being a major contributor to success [16]. Despite this, most languages are not interoperable [16, 17, 47]. Investigating when and why developers may fork a programming language file requires consideration of language needs and developer motivation. For example, some developers may fork a programming language because the original language has been combined with another new programming language. Other developers may fork a programming language to add or amend features to the language or to a subset of the original programming language.

A file repository fork is defined as an original repository where source code developers contribute sufficiently to benefit the OSS community [7].

As such, we need to understand developer forking motivation. There are currently a broad range of perspectives on OS developer motivation, ranging from individual to communities, and fork consequences on projects and organisations.

Although a variety of research methods have been adopted to predict OSS popularity, sustainability and survivability [3, 5-7, 9, 13, 16, 26, 34, 36, 42, 44, 46, 48], these methods are less useful for predicting programming language forking survivability. These research methods include surveys, interviews, content analysis and empirical studies that are subjective and potentially biased. For example, data samples were not large, reducing accuracy; data analyses and interpretation could be subjective or biased; and the study designs were unable to handle large datasets. In contrast, machine learning techniques work effectively with an abundance of data to leverage for training and testing.

2.7.1.3 Project forking

According to Nyman and Mikkonen's definition [3], a project fork is defined as source code copied from software, with the copy version used by the fork developers to develop their work. In other words, the piece of source code that is forked is an independent version, separate from the original source. Nyman and Mikkonen [3] looked at forking behaviour in the context of forked project survivability, quantifying project forking as the number of original projects forked by developers and comparing the number of original projects versus forked projects in GitHub. Many researchers seek to understand how forking impacts an original forked project and Nyman and Mikkonen's study [3] provided real-life examples of current high-profile OS projects that either started from a fork or were common targets for forking.

2.7.1.4 Software forking

Ikuine and Fujita [6] defined software forking as the continuous development of software, by the original developer or others. An original developer must share the source code

when other developers take over. Software forking focuses on the product itself, such as Microsoft software, Facebook software and email applications.

2.7.1.5 Social forking

Fung, Aurum and Tang [34] defined social forking in their study of nine JavaScript development communities in GitHub, with the highest amount of forks to identify the relationships within them and study how forks are used to facilitate OSS development. In their analysis, almost 7000 developers made approximately 8000 forks in different communities, with the most active developers making contributions to multiple communities. Their research indicated that forks are actively used by the development community to fix defects and to experiment with new features. What separates these forks from normal branching is that the changes do not necessarily need to be promoted to the original project upstream and can live in a separate fork that can still take any changes and improvements from the original project as updates. What separates a fork from a branch even more is that a fork can originate from either a subset of the forked predecessor's artifacts or from multiple predecessors' artifacts. A branch in turn is a copy of all the predecessor's artifacts [34].

2.7.1.6 Code forking

Code forking is defined as a forked project copied from an existing code base and moved away from the original project leadership direction. While addressing new requirements, code forking enables contributors or developers to add existing functionality. Despite its flexibility, there are developer community concerns, including forking maintenance, evolution and social factors. Another definition of a code fork is when a piece of source

code is downloaded or copied by a developer from an existing program which has the original version of the source code [3].

2.7.1.7 Programming language forking

Chua [26] examined language forking from the perspective of programming language adoption by project owners, finding three projects where Apache, Mozilla and Ubuntu Javascript languages were actively forked by developers. Chua and Zhang [27] then proposed three forking pattern types (‘once-only’, intermittent or steady) and potential reasons behind short-lived programming languages. According to [27], the definition of a programming language forking is an active and sustainable programming language that is adopted by developers or project owners and forked voluntarily by developers.

2.7.1.8 File repository forking

A file repository fork is defined as an original repository where source code developers contribute sufficiently to benefit the OSS community [7]. Developers who fork an original file repository can modify it for correcting bugs or adding new features, submitting bug fixes, adding new features, submitting pull requests and archiving copies. A repository written by a developer in their own programming language that is liked by other developers is also highly likely to be forked.

Table 2. 4: *Fork categorisation, sustainability and lessons learnt*

Type	**Interpretation**	**Studies**	**Citing authors within paper set**
Forking motivation			
Coding for revising requirements	Requirement change	Ernst et al., 2010 [40]	Ernst et al., 2010 [40] cited by Fung et al., 2012 [34]; Jiang et al., 2017 [7]
Seeking a coding job	Recruitment of contributors	Biazzini & Baudry, 2014 [42]	Nil
Licensing compliance	Licensing compliance	Biazzini & Baudry, 2014 [42]; Dabbish et al., 2012 [36]; Jiang et al., 2017 [7]	Nil
Software compliance	Software interoperability	Von Krogh et al., 2012 [9]; Meyerovich & Rabkin, 2013 [1]; Nyman, 2014 [46]; Tegawendé et al., 2013 [16], Jiang et al., 2017 [7]	Nil
Reviving original project development duration	Cessation of original project	Nyman, 2014 [46]; Robles & Gonzalez- Barahona, 2012 [5]; Ray & Kim, 2012 [47]; Tegawendé et al., 2013 [16]; Chua, 2017 [26]	Nyman, 2014 [46] cited by Jiang et al., 2017 [7]
Extending community social coding development	More community driven development	Dabbish et al., 2012 [36]	Ray et al., 2014 [17] cited by Jiang et al., 2017 [7]

Type	Interpretation	Studies	Citing authors within paper set
Ownership implication	Legal implication on ownership and conflict over brand ownership	Fung et al., 2012 [34]; Nyman, 2014 [46]; Nyman & Mikkonen, 2011 [3]; Ray & Kim, 2012 [47]	
Business strategy risk	Commercial strategy forks	Dabbish et al., 2012 [36]	
Team coding skill inequality	Differences among developer team	Nyman, 2014 [46]	
Community socialisation	Building new community through social interaction, sharing and collaboration	Dabbish et al., 2012 [36]; Fung et al., 2012 [34]; Robles & Gonzalez-Barahona, 2012 [5]	
Coding by socialising	Social network coding	Jiang et al., 2017 [7]; Fung et al., 2012 [34]	
Divergent specialisation	New specialisation, divergent technical views	Nyman, 2014 [46]; Nyman & Mikkonen, 2011 [3]; Ray & Kim, 2012 [47]	Nil
Objective misalignment	Different technical objectives		
Poor leadership	Poor project	Nyman, 2014 [46]; Nyman & Mikkonen, 2011 [3]; Robles &	

Type	Interpretation	Studies	Citing authors within paper set
	governance	Gonzalez-Barahona, 2012 [5]	
Culture trait	Cultural differences		
Software activity	Project specialty to generate commits	Ray & Kim, 2012 [47]; Tegawendé et al., 2013 [16]	Nil
Ecosystem	System between system sharing resources and infrastructure		
Forking sustainability			
Community activity	Communities retention	Ernst, et al., 2010 [40]; Gamalielesson & Lundell, 2013 [8]; Jiang et al., 2017 [7]; Nyman & Mikonnen, 2011 [3]; Azarbakht & Jensen [13]; Cosentino et al., 2017 [14]	Ray et al., 2014 [17] cited by Jiang et al., 2017 [7]; Gama-lielesson & Lundell, 2013 [8]
Forking lessons learnt			
No formal process	No guidance/ direction	Ikuine & Fujita, 2014 [6]; Fujita & Ikuine, 2014 [43]; Azarbakht & Jensen, 2017 [13]	Nil
Legal implication	Copyright	Glass, 2003 [15]; Azarbakht & Jensen, 2017 [13]	
	Licensing conflict	Moen, 2016 [18]; Azarbakht & Jensen, 2017 [13]	
Transfership	Project ownership	Ikuine & Fujita, 2014 [6]; Fujita & Ikuine, 2014 [43]; Cosentino et al., 2017 [14]; Azarbakht & Jensen, 2017 [13]	

Type	Interpretation	Studies	Citing authors within paper set
Product expertise shortage	Technical developers become product expert	Neville-Neil, 2011 [45]	
Upgrade of developer role to product role	Role movement	Glass, 2003 [15]; Ikuine & Fujita, 2014 [6]; Cosentino et al., 2017 [14]	Glass, 2003 [15] cited by Fung et al., 2012 [34]
Community divide	Divide community fork	Azarbakht & Jensen, 2017 [13]; Cosentino et al., 2017 [14]	Nil

2.7.2 What were the most popular methodologies used by forking researchers from 1990 to 2017?

Error! Reference source not found. presents data relating to methodologies across the 21 papers after they were carefully reviewed for study type, research methodology and data collection methods and type. Thirteen of the 21 papers were qualitative with data collection methods including stratified sampling (n=8), systematic study (n=5), qualitative interview (n=2), qualitative case study (n=2), survey and interview (n=1), stratified sampling and survey (n=2) and qualitative interview and survey (n=1).

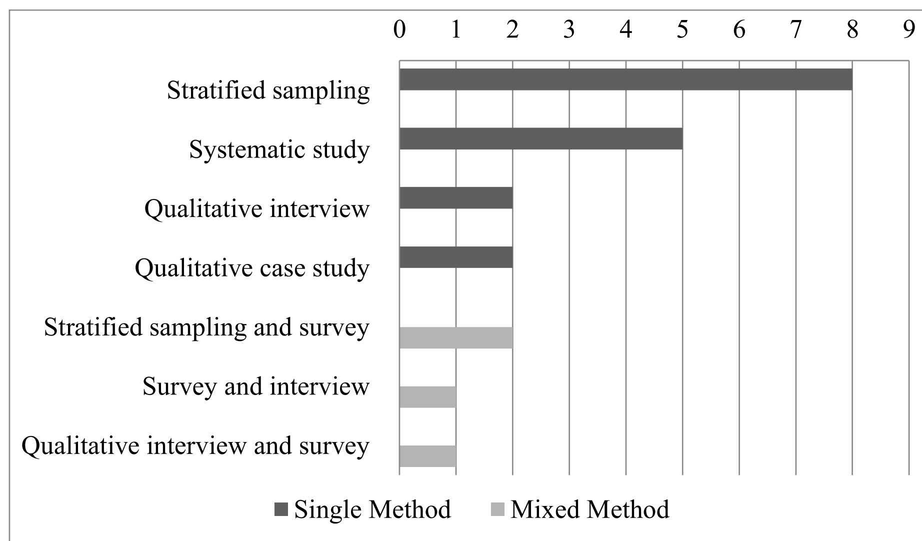

Figure 2.3: *Data collection methods in the 21 papers*

2.7.3 What aspects of open source forking have been researched and reported?

Error! Reference source not found. shows the units of analysis used in the 21 papers. In seven papers this was a comparison between non-forking and forking projects. Of the remaining 14 papers, six papers focused on the forking relationship on software releases, version control files and file repository and eight focused on OS project interactions with components, such as popular programming languages, the product and the successful

system, and analysing forking behaviour between the manager, developer, and end user (GitHub versus non-GitHub).

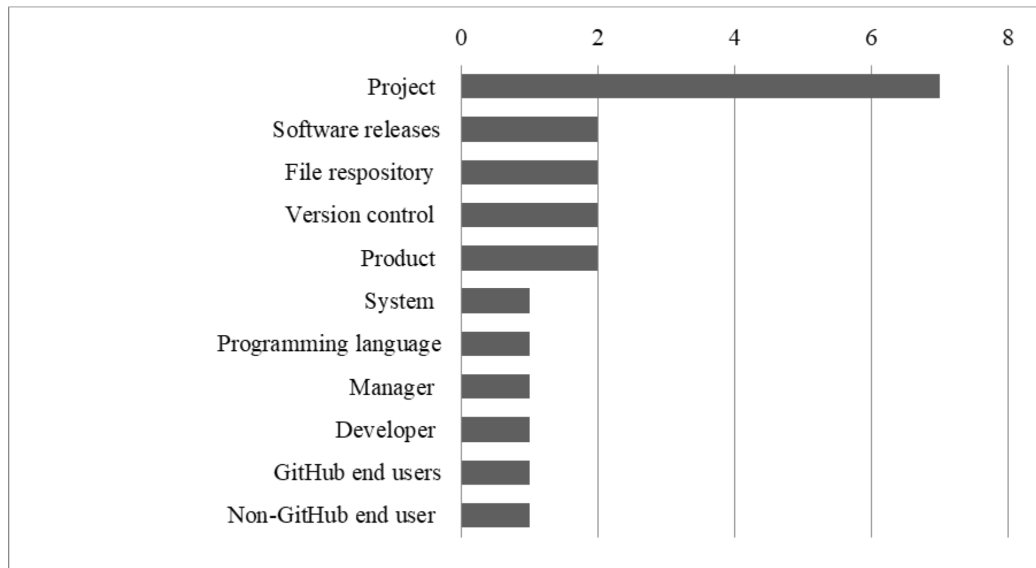

Figure 2. 4: Units of analysis in the 21 papers

Error! Reference source not found. shows eight types of forking lessons learnt on project compatibility issues that were identified in the 21 papers. In order of decreasing frequency of reporting, these were: no project ownership (n=4); no project guidance and the developer role becoming specialised (n=3); copyright, licensing and the software less likely to become proprietary, and a split community (all n=2 each). There was also one paper on losing developers as technical developers become product experts.

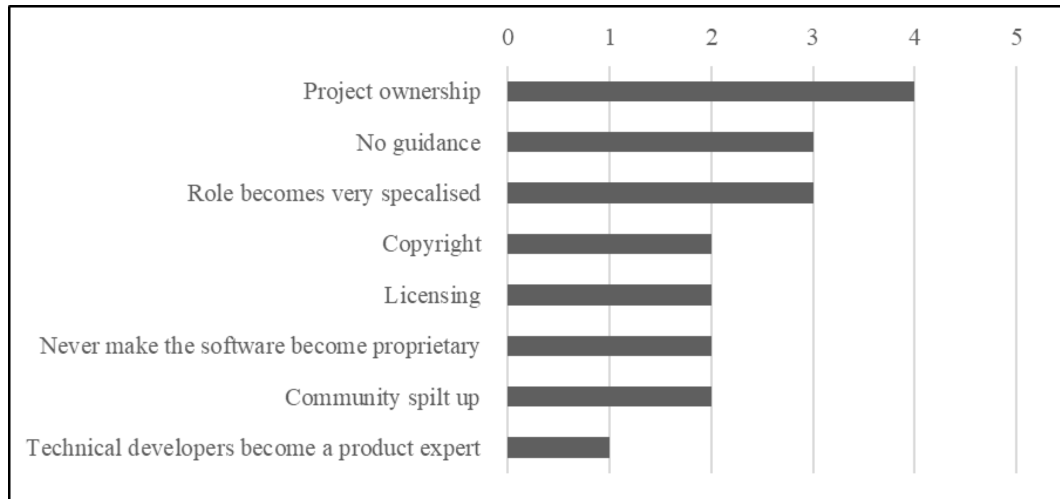

Figure 2. 5: Forking lessons learnt across the 21 papers

2.7.4 Newcomers or new developers forking motivation from 2020 to 2021

From 2017 to 2021, we reviewed other papers on forking motivation and could not find new forking motivational variable/s or reasons except for some papers that investigated OS newcomers' forking motivation. We examined this literature closely to evaluate motivations for first-time developers.

Subramanian et al. [56] analysed and investigated 3501 first-time developers who forked OS projects for the following core motivations: documentation, feature, bugs, refactoring, GIT-related issues and test cases. They commented first-time developers fork a repository to: 1) update documentation changes on files such as READMEs and/or explanatory comments; 2) add a new feature or a new functionality onto the project; 3) fix unexpected bug behaviour in code; and 4) make code more readable and understandable by refactoring it and conforming to coding standards.

What attracted to first-time developers to fork based on top three reasons are 1) editing documents, 2) adding or modifying new features and 3) correcting bugs. Instead of fixing

major features, the first-time developers contributed to fixing minor feature changes and bug fixing, for examples, they can fix an unexpected behaviour in the source code, memory allocation errors, concurrent tasks errors and requirements. inconsistency for instance, misspellings and assigned values mistakenly etc.

For first-time developers to complete these tasks, it is important to ensure OS environmental settings are properly configured, including the OS project is aligned with new or emerging technology as far as possible, a sustainable programming language is used and the project has a legitimate OS licence.

In this paper, the researchers mentioned they studied first-time developers who forked projects from The Apache Foundation (ASF) in GitHub [57] is to provide their support and services and support in ASF activities. Having the OS environmental setting configured can motivate first-time developers to fork and find the forking process enjoyable.

2.7.5 Shifting motivation through time and journey

The work of Gerosa et al. [58] reports shifting motivations through time and the shifting motivations through the 'self-journey' of developers. They refer to the shifting motivation through time as some developers' motivations contribute to OSS as a test of time from learning, fun and knowledge, sharing at the 'curiosity' period on every first-time OSS developers fork. After the curiosity period is over, developers shift their motivation focus onto social aspects, such as altruism, kinship and reputation. Shifting motivation through self-journey means that developers first contribute to OSS based on extrinsic motivations – such as ideology, own-use or education-related programs – and consequently shift to intrinsic reasons such as fun, altruism, reputation, and kinship.

2.7.6 Shifting forking motivation

We evaluated how first-time and non-first-time developers' forking motivation diminished, not only for personal reasons (figure 2.6). An absence of a good OS environment compliance can also affect both groups. A complete OS environment compliance is very important because it brings convenience and flexibility to them. It gives them confidence to explore, learn and contribute in a secure and safe yet challenging learning space. A highly desirable OS infrastructure environment allows developers to have the space to learn, grow and develop source code. A less desirable OS infrastructure environment has a negative effect on their motivation, especially when they face environmental barriers like using a less sustainable programming language, which may not provide benefits to them. They choose not to fork other repositories as often as they prefer. Similarly, a less useful technology and a highly restrictive OS licence would create multiple barriers to forking and increase the chance for first-time and non-first-time developers to stay away from the forking environment.

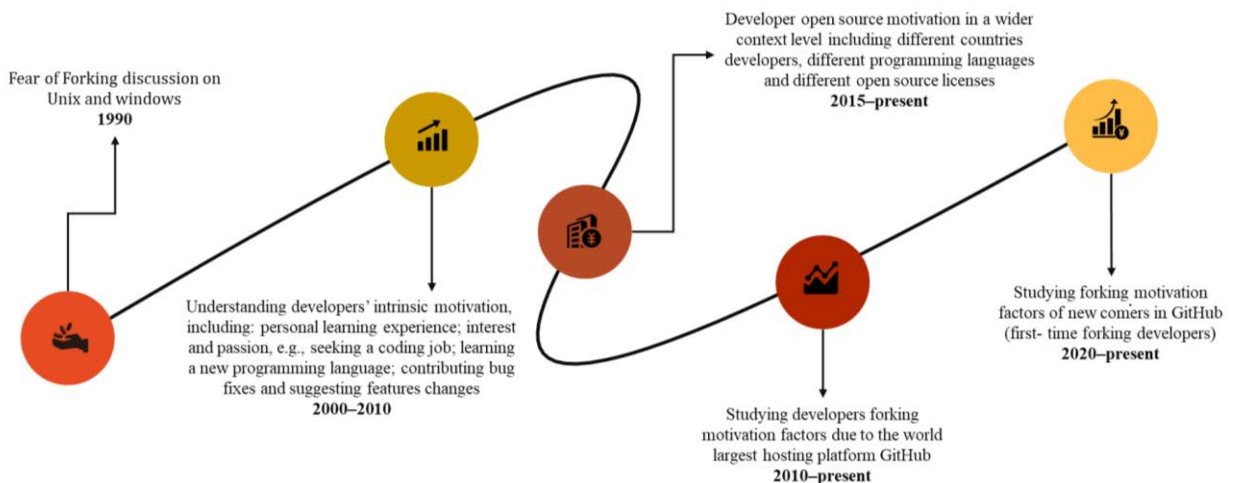

Figure 2. 6: The open source developers' motivation movement

2.8 Summary from the literature survey

Forking is one of the most critical techniques in OS research today. Our analysis of 21 papers from the first part of literature survey can help the OS community – educators, academicians, developers, project investors – to improve awareness of forking as a sustainable way to revive project health. The categories of forking lessons learnt highlight that forking consequences are likely to continue and remain a survival challenge to OSS developers. For example, if forking life span becomes short-lived developers could close a project or terminate the file repository.

According to our investigation, there is no research discussing how a lack of sustainable programming languages could reduce forking sustainability and viability. Programming language attractiveness drives and motivates developer desire to fork, helping to maintain forking health and activity. The usefulness of a programming language is the likelihood a fork can be generated effectively by developers. We strongly believe it is important to investigate how competitive programming languages can impact forking sustainability and to seek ways to prevent low forking performance, if necessary.

The following outlines our findings from the literature survey from Part I:

1. **Append new findings into the body of knowledge on OS forking behaviour**. Applying the combined approaches of SLR and Content Analysis revealed seven forking types interpreted by academic researchers and the latest interpretation found is file language repository fork. This novel insight will assist researchers on how forking is presented and interpreted and industry practitioners in reviewing project forking health, especially projects with programming language file repositories that are less adopted or forked by developers.

2. **Understanding forking consequences**. Case studies are an important way to highlight lessons learnt by researchers. This paper identified forking impacts and consequences, with one of the worst impacts being a political strategy that divides a project community and forms a new community. Forming a new community results in less contributions by developers to the original file repository, bug fixes or feature enhancement. Allowing accumulated bugs and feature enhancements to remain unfixed for a period of time can affect project health risk.

3. **More research is required on forking sustainability**. Reviewing these 21 papers revealed the importance of forking sustainability investigation as a top priority with two specific areas of interest.

    a. *Analysing forking from a social community perspective*. For instance, Azarbakht and Jensen [13] conducted a study to determine what motivates people to decide to fork (break away) in complex software development networks, and the type of changes a community needs to consider when deciding to divide [13]. Differences or conflicts on team communication, goals, styles or values can positively or negatively influence community interactions.

    b. Understanding the relationship between programming languages, repositories and developer forking interest to more accurately predict OSS forking motivation and behaviour.

4. **Studying forking sustainability using a SLR for software development with GitHub**. Consentino et al. [14] used a SLR to show that project longevity and forking chances are the two highly dependent variables on the project. They also discovered that developers provide additional contact information (e.g., email address, personal website URLs that are clearly active) to increase social interactions between a project owner and forker [14]. Future work could include developing a prediction model for

fork effectiveness from forking motivation classifications in response to language repository files, where programming language survival time is critical to an OS projects' health and survivability.

Chapter 3: Literature Survey Research Methodology

3.1 Overview

This chapter reviews the research methodologies used to investigate OSS, classified into methodologies based in literature surveys, data mining or machine learning. As this work is yet to be published the chapter is formatted in a more traditional style.

3.2 Motivation

The primary motivation in conducting the research methodology survey was to achieve a holistic understanding of the types of research methods that have been adopted by researchers previously to investigate OSS variables, and to contribute a paper to the field describing these different methods. We also aimed to affirm our choice of using a predictive learning algorithm, KNN, to address our research problem on forking scarcity and low forking visibility to understand developer forking motivation and behaviour.

3.3 Introduction

GitHub is a social software development platform that is widely used by developers to collaborate on OS projects. Since its inception, the number of GitHub users, developers, projects and repositories have all increased significantly: as of January 2020, the GitHub user population was over 40 million users, with over 190 million repositories, of which 28 million were public [59].

Improving GitHub's hosting platform quality requires investigation of the factors that may impact it. For example, topics that could be explored include the motivational behaviour of newcomers, the sustainability of long-term contributors, social network success ability, forking longevity, the reliability of a software repository, bug defects and

fixing resolutions, sustainable programming languages, and/or OS licences that meet full compliance.

There is extensive research on GitHub; however, very little attention is paid to research methods employed in OSS. Selecting a proper research method is important in addressing the research problem in the appropriate way to provide meaningful data that can be interpreted accurately by others and applied to their context if useful. We aimed to fill this academic gap.

3.4 Literature Survey Selection Criteria and Categorisation

We searched academic databases using the following keywords: "Open Source Survey", "Open Source Forking", "Open Source Prediction" and "Open Source GitHub". We then applied the following criteria to the search results (

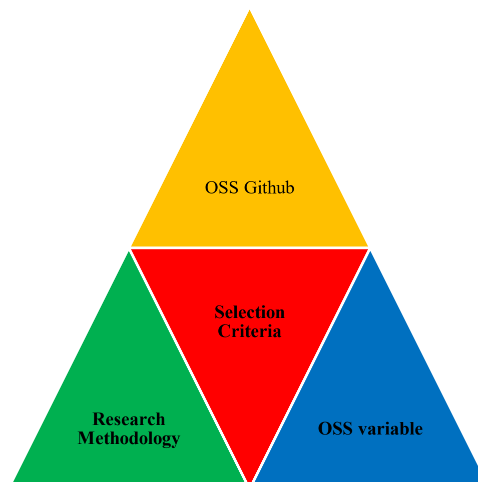

Figure 3. 1):

1. The research study must be in OS GitHub.
2. The research article must explain a research methodology and how it was applied.
3. The discussion of an OSS variable in the research article must be relevant to our research context addressing the forking topic or a function of the OSS infrastructure

of concern to developers, users, newcomers, OS licence, programming languages, file repositories, business process models, fork, etc.

4. The research article must be high quality and peer reviewed.

5. The research article must have been published between 2003 and 2021.

Research articles that met these criteria were scrutinised. The abstract was reviewed for OSS variables and descriptions of research methodologies were reviewed to ensure it related to OS and GitHub.

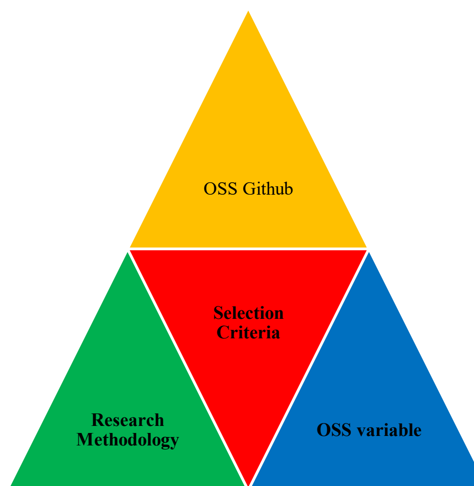

Figure 3. 1: Paper selection criteria

After reviewing the research articles, they were categorised as being based in:

1. **Surveys:** Articles that focused on understanding OS variables previously investigated by other researchers in disciplines such as Information Systems, Social Science, Software Engineering, Information Technology and Computer Science.

2. **Data Mining Algorithms:** Articles that focused on understanding which type of OS variables were mined, identified and detected.

3. **Machine Learning Algorithms:** Articles that focused on a specific OS variable that was mined, identified, and outcomes predicted.

3.5 Category I: Survey-based Research Methodology

Ten literature survey research papers that we reviewed discussed OS variables related either to forking topics or a function of the OSS infrastructure (Table 3. 1). Three of the eight (37.5%) survey papers targeted roles such as developers, users, promoters and newcomers. The objective of conducting a literature survey for three researchers [7, 11, 60] was to seek clarity and interpretations of roles.

Three papers adopted two research methods to examine GitHub users. Jiang et al. [7] applied a survey method and a regression analysis on 236,344 developers and 1,841,324 forks to categorise developers into three groups. The first group is 1) bug maintenance developers they forked repositories to solve issues like pull requests, fix bugs, add new features; 2) the second group is programming language developers who are more likely to fork a repository if a programming language is written familiarize to them and 3) the third group of forked developers who fork repositories from their project owners. Celińska [60] applied a logit model and descriptive statistics to classify 3,915,138 GitHub users into three groups: those who used popular programming languages; those who have a good reputation within the community; and those who provided additional information to attract more developers to join GitHub. Lastly, Balali et al. [11] used a SLR to identify barriers to using GitHub then interviewed 10 newcomers about their experiences of barriers while using GitHub. From the interviews, they found 34 of the 44 identified barriers can affect newcomers, while only 19 can affect mentors.

Four research articles examined software repositories as an OS variable [61-64]. Software repositories is one of the features of GitHub and is an essential function for OS

infrastructure. Teng et al. [63] were interested in social networks as there is little research on software repositories and social network and the interaction between them. They conducted a literature survey and discussed five types of mined software repositories which they are 1) email archive; 2) online forked communities; 3) blogs; 4) bug tracking systems; and 5) version control – to determine how OS communities interact. They then studied their behaviour patterns and concluded that the social networks, blogs and online communities have gained popularity and further research in this area looks promising. Similarly, Zhang et al. [64] applied Teng et al.'s categories to the results of a literature survey of 20 papers that emphasised developer social network construction, analysis and construction importance when studying developers social network patterns.

The additional contribution from Siddiqui and Ahmad [62] on software repositories survey is Deployment Logs (DL) and code repositories. They surveyed and compared Mining Software Repository (MSR) tools used in six major open source projects: Dynamine (DME); SoftChange (SCG); Chianti (CNI); Hipikat (HKT); Kenyon (KYN); and Apfel (AFL). Their metrics included different parameters under several categories: the user dimension group included manager, developer, user and tester; time included past, present and future; information source included Control Version System (CVS), issue tracking and software release systems. They found all six OS projects used the CVS system while for the other metrics the support systems varied across projects, e.g., change management, defect tracking, archive mailing lists, infrastructure requirement, online/offline, storage required, input data and language dependency. Borges et al. [61] applied multiple linear regressions to 4,248 software repositories starred by developers to predict popularity and successability. However, they did not discuss starring of forked repositories.

Hora and Valente [65] studied the factors that impacted GitHub repository popularity. They downloaded a GitHub dataset that consisted of rated stars by developers on 2279 popular GitHub repositories and found the star rating on programming languages affects project popularity. The programming language JavaScript scored 3697 stars, versus the programming language Go with 3549 stars and the non-programming language HTML with 3513 stars. The lowest star ranking of the three programming language repositories was PHP (3245), Java (3224) and Python (3099). They identified the application domain as another important factor. OS files such as systems software, web libraries, frameworks and documentation rated more stars on application domains, and organisation repositories were more popular than individuals' repositories.

In contrast, Do et al. [22] proposed a new way for fork detection and duplication residing in repositories. They downloaded more than 3 million software repositories and determined that 52,484 were active, 8434 were forked and 7441 were release-based forked repositories. Their preliminary investigation revealed a significance relationship between forking patterns and fork success indicators.

A more recent OS variable of interest is communities' code of conduct. Li et al. [66] applied a qualitative analysis approach on a small sample size of GitHub issues concerning whether the code of conduct had been used positively or negatively to address project issues. They downloaded 52,000 public GitHub repositories and found 50,000 were the most popular (based on star ratings) and the remaining 2000 were not rated. 6,566 (12.6%) of the repositories from the project root had a code of conduct.

Table 3. 1: Literature Survey Research Methodology in OSS

No	Category/ Cluster	OSS Variable	Classification	Reported Size	Dataset	Non-Machine Learning Technique	Paper
1	Role	Forking developer	1) Bug maintenance developers forked repositories to solve issues like pull requests, fix bugs, add new features; 2) Programming language developers were more likely to fork a repository if a programming language is written in a familiar language; and 3) Forked developers who fork repositories from their project owners	236,344 developers, 1,841,324 forks	GitHub	Literature survey, statistical regression	Why and how developers fork what from whom in GitHub 2017 Software engineering. 2017. [7]

No	Category/ Cluster	OSS Variable	Classification	Reported Size	Dataset	Non-Machine Learning Technique	Paper
2	Role	GitHub user	Programming language users	3,915,138 users	GitHub Torrent	Logit model, descriptive statistics	Coding together in a social network: collaboration among GitHub users. 2018. [60]
3	Role	Newcomers	Newcomers' barriers	Two women and eight men who are experienced OSS Five of them had industry closed-source projects, and one of them had experience in OSS and academia	GitHub,	Systematic survey, interview	Newcomers' Barriers. . . is that all? An analysis of mentors' and newcomers' barriers in OSS projects. 2018. [11]
4	Social network	Software repositories	Five Software repositories classification	1,251 software repositories	GitHub	Literature review	A survey of mining software repositories in social network. 2020. [63]

No	Category/ Cluster	OSS Variable	Classification	Reported Size	Dataset	Non-Machine Learning Technique	Paper
5	Developer social networks	Construction, analysis, and applications	Social interaction from developers regarding information related to software development, construction, analysis and projects.	104 papers	GitHub	Literature review	Developer social networks in software engineering: construction, analysis, and applications. 2014. [64]
6	Software repositories:	Mining software repositories	Run time repositories, code repositories, fault prediction	9 papers	GitHub	Literature review	Data mining tools and techniques for mining software repositories: A systematic review. 2017. [62]
7	GitHub stars	Star rating on each GitHub repository	Star popularity	4248 repositories	GitHub	Multiple linear regressions, KSC clustering algorithm	Predicting the popularity of GitHub repositories. 2016. [61]
8	Main factors that impact	stars, including programming	Factors	2279 repositories	GitHub	Descriptive statistics	Understanding the factors that impact the

No	Category/ Cluster	OSS Variable	Classification	Reported Size	Dataset	Non-Machine Learning Technique	Paper
	GitHub project stars, including programming language, application domain	language, application domain					popularity of GitHub repositories. 2016. [65]
9	Software repositories	To detect fork and duplicate repositories	Fork and duplicate repositories	3 million software repositories	GitHub	Descriptive statistics	Mining and creating a software repositories dataset. 2020. [22]
10	Code of conduct conversation	Codebook development, reliability, and application.	Code of conduct conversation detected on repositories	6,566 responses on commenting the issues	GitHub API	Qualitative analysis	Code of conduct conversations in open source software projects on GitHub. 2021. [66]

3.6 Category II: Data Mining Algorithm-based Research Methodology

This section summarises ten research articles that used data mining methodologies (Table 3. 2). These articles aimed to investigate: 1) OS licence changes; 2) business process management notation methods and tools used in GitHub; 3) source code changes in OS projects; 4) change analysis activity patterns; 5) malware activity in source codes; 6) identifying programming languages that can improve productivity and quality; 7) identifying programming languages via images; 8) how communication may serve a function on the bug fixing activity between developers; 9) the communicative activity on forking and 10) fork types at windows and network level. Each article discussed the purpose of using a data mining method for analysis, either text-based, data-based, pattern-based or sentiment-based. Other OS variables are also important to consider for mining purposes as its aim is to identify patterns or classifications.

These papers were selected as part of the literature review to understand the current trends on OSS variables that being investigated. All ten articles are highly relevant to our work on predicting forking performance, with the OS infrastructure playing a pivotal role in predicting forking patterns and performance that align with developer motivational behaviour. OS infrastructure consists of OS licences, programming languages, developers, software repositories, business process methods and models, bug fixing activities, fork mechanism, etc.

From our knowledge, the field of forking performance prediction research is small but growing. Six of the 10 papers were published very recently, in 2020 or 2021, and these studies are very helpful in informing choice of research method.

Vendome et al. [67] mined and downloaded 16,221 projects written in a Java programming language hosted on GitHub and analysed licence changes in 1,731,828

commits. They classified the licence changes into three patterns: 1) from no licence to some licence (N2L); 2) from some licence to no licence (L2N); and 3) from some licences to some licences (L2L).

Heinze et al. [68] mined 6,163,217 repositories and identified 1,251 repositories of Business Process Modelling Notation (BPMN) artifact. They determined the total artifacts across four types of files: 16,907 artifacts on XML BPMN 2.0; 384 XML artifacts; 1635 Image file artifacts; and 2380 artifacts from other files. They analysed and studied how BPMN process model artifacts distributed across geographical locations and found China has the most BPM contributors, followed by Germany then the US. They also discovered a high percentage of BPM artifact duplications.

The research paper entitled "Predicting source code changes by mining change history" by Ying et al. [69] discussed the identification of patterns on source code changes by applying data mining techniques. Ying et al. [69] evaluated two OS projects – Eclipse and Mozilla – from GitHub and evaluated the predictability. They applied association rule mining algorithms [70, 71] to determine the frequency occurrence of a set of source files changing patterns in the database. They also applied pattern mining [70, 71] to detect the source code change history pattern. They categorised Mozilla and Eclipse by interestingness value as surprising, neutral or obvious, according to how well the files were structured so that developers were notified. The Mozilla project had two surprising recommendations, two neutral and five obvious, compared with the Eclipse project, which had one surprising, two neutral and seven obvious. The two 'surprising' Mozilla projects were categorised based on one "cross-language" case (file dependencies written in different programming languages were not easily found by developers) and a source code duplication case, where the code was generated from a number of source code modifications.

Saini et al. [72] performed change evolutionary patterns on 106 OSS projects over a time period of 76–81 months to study the types of changes. They applied a keyword-based classifier technique on change messages and categorised them into corrective, adaptive, perfective, preventive or enhancement. Next, they adopted a cluster analysis technique to detect hidden distinctive change patterns for each change type range from low, moderate and high. They classified a high and moderate activity project has the highest number of changes over the project lifetime. However, their results unfortunately did not show significant correlation between change types and domains or languages in their projects.

La Cholter [73] mined 1,835 repositories to study malware-related files as source code written in either C or C++ in a window environment has been targeted frequently by malware.They classified malware as benign, suspicious or malicious. The highest malware pattern was benign (20,060), followed by suspicious (4335). System files from Win32/64, DLLs, EXEs and PreWind32 were then mined and, unsurprisingly, Win32/64 was found to contain the highest benign malware. Their detection Virus Total (VT) test showed 1353 files with malware and 9060 files without malware.

One of the decisions made by developers when forking a software repository related to the programming language is that if a developer is familiar with the programming language used by the project owner, he or she is likely to fork the repository and contribute. Altherwi et al. [74] predicted software development productivity and quality by comparing scripting languages and traditionally compiled system programming languages. They mined a population of 15,000 GitHub projects from a five-year period (January 2012 to December 2017) and identified eleven programming languages in a sample of 4349 projects. JavaScript, Python, PHP and Ruby were classified as the four scripted programming languages and Java, Go, Objective-C, Swift, C#, C++ and C were classified as the seven system programming languages. They found that scripting

programming languages are more widely used than system programming languages, with evidence that JavaScript is the dominant scripting language, used in 2174 projects. Java dominated the system programming languages, found in 2154 projects. They concluded the programming language choice can affect the development process but did not have an opinion on which type of programming language can affect productivity, as other variables need to be considered, such as developer coding experience, skill and background, as well as the project type and development environment.

A group of researchers from Italy and France [75] developed a programming languages identification (PLI) technique. 149 programming languages were detected from their mining of 1000 repositories from 300,000 code snippet images. The researchers classified them into four types: alphabetic characters, digits, symbols and a combination of characters with substitutions of all non-blank characters. They evaluated the performance of these five classifications and found alphabetic characters and symbols for instance parenthesis, punctuation and mathematical operators have higher visual recognisability than digits and indentation.

Bug fixing is an active area in OS development. Ramírez-Mora et al. [76] looked into three tracking systems and extracted over 500,000 comments and 89,000 bugs from a hundred OSS projects. They found a significant difference on the distribution of comments across Apache, Red Hat and Spring on fixed and not fixed issues. The other important finding was a higher rate of emotional and emotive comments from developers when dealing with bug resolutions or for bugs that took a long time to implement.

In a slightly different approach, Brisson et al. [77] studied 385 software families of 13,431 software repositories to understand how developers communicate. They found out that projects in the same family (forking) of multiple repositories showed a positive relationship between the fork volume and the number of users who contributed. Further,

the number of repository stars showed a positive relationship with communication for fixing bugs by followers who were outside the family.

Interestingly, Pietri et al. [78] examined the structure and size of fork networks to better understand forking. They classified forks as forged, committed or shared roots. They found 18.5 million fork repositories and 25.3 million network repositories were forge forks, 20.1 million fork repositories and 24 million network repositories were commit forks, and 25.3 million fork repositories and 18.5 million network repositories were shared root forks. Their evidence also suggested that what developers typically recognise as “forks” are the share commit forks.

Table 3. 2: Data Mining algorithm-based type research methodology

No.	Category	OSS Variable	Classification	Reported Size	Dataset	Data Mining Technique	Paper
1	Component	Licence	OS licence changes	Licence changes in 1,731,828 commits, 16,221 GitHub Java projects	GitHub	Code analysis, data mining	Licence usage and changes: A large-scale study on GitHub. 2017. [67]
2	Component	BPMN process model artifacts	xml	6,163,217 repositories	GitHub, GitHub Torrent, GitHub API	Google query	Mining BPMN processes on GitHub for tool validation and development. 2020. [68]
3	Source code development	Source code changes	Eclipse and Mozilla project	Changes to >20,000 files, >100,000 versions of source files	GitHub	Association rule mining algorithm	Predicting source code Changes by Mining Change History. 2003. [69]
4	Component	OS software change	Change classification, corrective, adaptive, perfective, preventative,	GitHub recorded message changes from 106 OSS projects	GitHub	Cluster analysis K-medoids algorithm	Change profile analysis of open-source software systems to understand

No.	Category	OSS Variable	Classification	Reported Size	Dataset	Data Mining Technique	Paper
			change cluster: high, moderate or low activity				their evolutionary behaviour. 2017. [72]
5	Software or code threat	Malware written in C or C++	Malware written in C or C++	1835 repositories	GitHub, Git	Virus total query	Windows malware binaries in C/C++ GitHub repositories: Prevalence and lessons learned. 2021. [73]
6	Component programming languages	Programming languages productivity and quality	software development productivity and quality	Mined 15,000 projects, including 4349 sample projects.	GitHub	Data mining, statistical analysis	Assessing programming language impact on software development productivity based on mining OSS repositories. 2019. [74]
7	Component: programming language	Programming language identification	Scrambling alphabetic characters, digits, symbols, combinations and substitution of non-blank characters	code snippet of 300,000 images, 149 programming languages	GitHub	K snippets convolutional neural networks (CNNs),	Image-based many-language programming language identification. 2021. [75]

No.	Category	OSS Variable	Classification	Reported Size	Dataset	Data Mining Technique	Paper
8	Software code development	Communication function	Fixed and not fixed bugs, comments classification under projects, Apache, Red Hat, Spring	>500,000 comments, 89,000 bugs from 100 OSS projects	Issue tracking system from: Apache's JIRA, Red Hat and Spring	Data analysis	Exploring the communication functions of comments during bug fixing in open source software projects. 2021. [76]
9	Communication	Commun-ication within software repositories and forks	Communications within software repositories and forks	385 software families, 13,431 software repositories	GitHub	Sentiment analysis	We are family: Analyzing communication in GitHub software repositories and their forks. 2020. [77]
10	Fork classification	Fork classif-ication based on activities	Forge forks, commit forks, shared root forks	Software Heritage Graph Dataset and GHTorrent of 71.9M repositories	GitHub projects	Fork network algorithm	Forking without clicking: On how to identify software repository forks. 2020. [78]

3.7 Category III: Machine Learning Algorithm-based Research Methodology

In this section we explore machine learning methodology in the context of OSS to investigate which methods are used in OSS research, and how. No doubt the machine learning technique is promising, but not all articles published on machine learning are suitable for our analysis. To be considered relevant to our study, articles must satisfy two criteria: the OS variables must be associated with the OS environmental infrastructure variables and the database must be GitHub. In total, six relevant papers were reviewed as relevant and useful (Table 3. 3). Role, issue and source code were the three OS variables that we found using more than one type of machine learning methods. Here we compare each machine learning method used in the same group; for instance, a variety of machine learning methods are introduced for role, with different OS variables for different machine learning methods.

Altogether four roles are used when applying machine learning methods for performance prediction: technical users [79], long-term contributors [80, 81], promoters [82] and newcomers [83]. Technical users include backend, frontend and full stack users, as well as mobile development and data science users. Montandon et al. [79] downloaded and analysed 2284 developer records then adopted stratified baseline, random forest and naïve Bayes. Competitive results obtained. Random forest achieved a high precision of 0.77 and naïve Bayes scored 0.62 for the recall result. In addition, their results showed programming languages were predominant across all five roles. To examine long-term contributors, Eluri, et al. [80, 81] downloaded 917 projects and 75046 contributors from GitHub and grouped them into five dimensions. Next, they used five machine learning algorithms to determine which machine learning technique provided the most accurate result. Random forest performed the best on evaluating long-term contributors;

newcomers are long-term contributors who stay for a period of time in forking projects. Similarly, newcomers became long-term contributors after a period of use on a programming language with many commit submissions.

Du et al. [82] studied the third role, promoter, by identifying 1023 suspected promotion accounts from GitHub Archive from 2015 to 2019 then applying a SVM classifier to detect 63,872 suspected promotion accounts from all active users . They then analysed these accounts, showing GitHub promotion services were exploited by a group of small businesses to promote their products. They found normal accounts have, on average, 4.50 forks, which is lower than the suspected promotion accounts. Normal accounts have 21.17 stars and suspected promotion accounts have 91.54. This means the suspected promotion accounts have 1.98 times the fork operations and the 4.32 times the star operations.

The fourth role is newcomer. Fronchetti et al. [83] downloaded 450 GitHub repositories and applied the K-spectral centroid (KSC) clustering machine learning algorithm to investigate whether project age, number of stars, programming language used and the number of pull requests contributed to newcomers' growth rate.

They applied a random forest classifier to predict three patterns: logarithmic, exponential and linear growth. They then determined that time, such as the review of pull requests, project age and programming language contribute to newcomers' growth patterns.

The second OS variable is issue, examined using a ticket tagger machine learning algorithm. Kallis et al. [84] downloaded 30,000 issues and classified them either 1) a bug report, 2) a feature request or 3) a question. They compared evaluations using J48 versus ticket tagger and found metrics on ticket tagger were more accurate on bug, feature request and question. The precision and recall results were at least 20% higher than J48.

The third variable is source code on bug defects [85], with bug reports classified by programming language (Python, Java, C and C++). A total of 14,950 GitHub bug reports were evaluated by a number of machine learning algorithms – including linear SVM, (LSVM)], SVM and Nu-SVM (NSVM) – with radial, sigmoid and poly kernels, based on libSVM, Gaussian process (Gauss) classifier, and K Nearest Neighbour (KNN). Rokon et al. [86] investigated malware source code to understand malware behaviour and the techniques used to detect malware source code repositories. They downloaded 97,000 repositories and identified 7504 malware source codes. They then applied the natural processing language (NPL) machine learning algorithm to filter the naming convention. They evaluated fork, stars and watchers on repositories to determine which influence malware, findings that at least 100 fork repositories have a high influence compared with stars and watchers.

To understand the dynamics of GitHub, Zhou et al. [87] developed a tool called GitEvolve, which predicts GitHub repository evolution and the ways in which users interact with them. They used the deep neural network machine learning algorithm and developed a system that can predict when, where and what user group will next interact with a given repository. A graphic representation learns to encode the relationship between repositories to better predict popularity.

Lastly, Weber et al. [88] investigated features that can differentiate between popular and non-popular Python projects on GitHub. They mined 2000 projects and identified 38 features, which they evaluated using a random forest classifier to predict current popularity. They discovered that, unlike non-popular projects, popular projects have in-code features that strongly signal more documentation and use the 'with' statement more frequently.

Table 3. 3: Machine learning research-based methodology in OSS

No.	Group	OSS Non-fork Variable	Classification	Reported Size	Dataset	Machine Learning Technique	Paper
1	Role	Technical user	System level: backend, frontend, Discipline level: full-stack, mobile development, data science	2284 developers	Stack exchange data explorer (SEDE)	Random forest and naïve Bayes	Mining the technical roles of GitHub users. 2020. [79]
2	Role	Long-term contributor	Project level: Activity level:	917 projects, 75,046 contributors	GitHub GH Torrent	Naïve Bayes, SVM, decision tree, KNN, random forest	A large-scale study of long-time contributor prediction for GitHub projects. 2021. [80]
3	Role	Long-term contributor	Long-term or non-long-term contributors	70,899 observations, 888 repositories, 56766 developers	GitHub	Naïve Bayes, KNN, logistic regression, decision tree, random forest	Predicting long-time contributors for GitHub projects using machine learning. 2021. [81]

No.	Group	OSS Non-fork Variable	Classification	Reported Size	Dataset	Machine Learning Technique	Paper
4	Role	Promoter	Promotion account: 1) hidden GitHub functionality; 2) small businesses exploit GitHub promotion services	63,872 suspicious promotion accounts (2015–2019)	GitHub archive	SVM classifier	Understanding promotion as a service on GitHub. 2020. [82]
5	Role	Newcomer	Mixed factors: project age, star number, program-ming language, text files to help contributors.	450 repositories	GitHub	KSC clustering algorithm	What attracts newcomers to onboard on OSS projects? TL;DR: Popularity. 2019. [83]
6	Issues	GitHub issues	Classify issue by topic title and description into bug report, feature request or question	30,000 issues	GitHub	Ticket tagger, text Mining	Predicting issue types on GitHub. 2021. [84]
7	Source code	Source code	Bug reports by	14,950 bug reports	GitHub	LSVM, SVM, NSVM	Estimating

No.	Group	OSS Non-fork Variable	Classification	Reported Size	Dataset	Machine Learning Technique	Paper
		defects	programming language (C, C++, Java, and Python)			with radial, sigmoid and poly kernels (based on libSVM, Gaussian process, KNN and) Classifier random forest and multi-layer perceptron classifiers	defectiveness of source code: A predictive model using GitHub content. 2018. [85]
8	Source Code	Malware source code	Malware IoT, Window, Linux phone	97,000 repositories, 7504 malware source codes	GitHub	Natural language processing	SourceFinder: Finding malware source code from publicly available repositories in GitHub. 2020. [86]
9	To predict GitHub repository evolution and different ways	Multitask architecture	Current model could be simplified to remove multitask output and be trained to predict		GitHub	Deep neural network, graphical representation learning	GitEvolve: Predicting the evolution of GitHub repositories. 2020. [87]

No.	Group	OSS Non-fork Variable	Classification	Reported Size	Dataset	Machine Learning Technique	Paper
	users interact with them		single specific popularity aspects, e.g., number of fork or watch events				
10	OSS popularity	Classify GitHub Python projects into Popular and non-popular	1000 projects	More users, more statement	GitHub	Random forest classifier	What makes an open source code popular on GitHub? 2014. [88]

3.8 Machine Learning: A K Nearest Neighbour Method

Silverman and Jones [89] describe how Fix and Hodges were the first group to introduce the K Nearest Neighbour – KNN – method in 1951. In 1967, Cover and Hart [90] then reviewed and refined the method. However, Cunningham and Delany [91] called KNN 'lazy' because the entire dataset does not require learning; there is no training time and the training data does not train itself. Instead, the training dataset memorised by itself.

The popularity of the non-parametric algorithm, KNN, increased because of its simplicity; it is easy to implement, easy to understand, effective and more accurate than many other classification algorithms [92-99]. It is one of the algorithms used in machine learning for a variety of applications required to solve a range of business problems, including unclassified and unpredictable. It has an ability to provide high accuracy, based on the fact that the prediction precision varies on the distance measured to determine similar features between observations. Unlike Cunningham and Delany [91], Jiang et al. [95] and McCord et al. [96] claimed that KNN is, in fact, a very successful and useful algorithm, especially as the calculation time is quick, data easy to interpret, and the algorithm has regression and classification versatility and high accuracy. Furthermore, there is no need to make any assumptions about data or build a model. In fact, it is far better than some other supervised learning algorithms, for instance, decision trees and the naïve Bayes classifier.

One of the critical success factors of KNN is that it uses mathematical calculations of distance metric from the similarity measure of features to determine the

"nearest neighbour". It also chooses the best parameter – K – which is found through cross-validation techniques. where K is dependent upon the data value.

3.8.1 Euclidean distance metric

There are nine distance metrics in KNN, with four of them widely adopted: Euclidean, Manhattan, Minkowski and Hamming distance [91] (Table 3. 4). All distance metrics calculate the distance dimension in a different way and, as a result, the output values differ.

Table 3. 4: Four widely adopted KNN distance metrics

Type	Concept	Distance Dimension
Euclidean	It is a straight-line distance between 2 real-valued vectors.	It calculates the shortest distance between two points
Manhattan	It calculates the distance between two data points in a grid-like path.	It calculates the sum of absolute differences between points across all the dimensions.
Minkowski	It is a generalized distance metric.	It calculates the sum of distance between two points in any two vector spaces (N dimensional real space).
Hamming	It is for comparing two binary data strings.	It calculates the string similarity between two strings of the same length.

3.8.2 Adopting Euclidean distance: characteristics identification and rationale

We chose to use the Euclidean distance metric after determining that its characteristics align best with our research study. We aimed to compare and determine fork visibility performance dissimilarity.

3.8.3 Identifying Euclidean distance characteristics

We identified four relevant characteristics:

- *Default metric:* often mentioned as the "default" distance used for measuring the similarity between observation and effective for identifying classification and K means clustering for finding the K nearest points of a specific point.
- *Dimension:* good at handling low dimensional data.
- *Real, successful applications:* Amazon and Netflix use this to recommend books and programs to watch based on previous customer behaviour.
- *Standardised variables:* This is necessary for variables in different measurement scales to balance the computation of distance effect. The Euclidean distance computed on standardised variables is called the standardised Euclidean distance

3.8.4 Our research dataset characteristics

The monthly fork data in the dataset can be large, difficult to analyse and interpret easily or correctly as they are quantitative count. Meaning to say, Monthly fork count ranging can be quite wide ranging from 100 to 10,000 per month. To calculate high fork visibility distance between repositories classification, Euclidean distance is strongly recommended.

We defined five characteristics of our dataset that would need to use Euclidean distance:

1. 108 related fork features were identified and used to predict healthy fork file repositories in response to developer motivation and behaviour.

2. A six-year period was chosen for the dataset (2015–2020), containing a large fluctuation in monthly fork count, ranging from tens to thousands. The variables were mostly binary.
3. We normalised fork count range from tens to thousands.
4. To calculate how close a distance point would be based on a similar feature of OS infrastructure compliance to predict healthy fork performance that can increase developer forking motivation and behaviour.
5. To calculate how far the distance point would be based on a different feature of OS infrastructure compliance to predict healthy fork performance that can increase developer forking motivation behaviour.

In the next chapter, we present the two models used in this study. One of them is a roadmap to summarise the literature review methods to investigate the two topics in Chapters 2 and 3. Chapters 5 and 6 provide detailed explanations of how we applied Euclidean distance to test our dataset.

Chapter 4: Models

4.1 Overview

This chapter introduces the two models used in this study. The first is a literature survey road map, showing the processes used to conduct the literature survey in Chapters 2 and Chapter 3 to cover both breadth and depth of literature and address the research problem.

The second model presented is the 'Chua and Zhang Predicting OSS Forking Pattern Model', which incorporates the KNN algorithm to predict high forking visibility. The model proposes detecting OSS patterns and predicting high fork visibility from repository classifications to interpret developer forking motivation and behaviour.

4.2 Literature Survey Road Map Model

The model shown in Figure 4. 1 outlines the two processes used in Chapters 2 and 3 to summarise the literature on OS variables and research methodologies. The first process aimed to identify OS forking variables to better understand OS developer forking motivation and behaviour; the second aimed to collate, compare and contrast existing research methodologies that were applicable to our research questions to inform our study design.

While reviewing the OSS forking literature, we realised that it is important to first understand and study the GitHub hosting platform and its features. By analysing GitHub features we can examine the way developers communicate between themselves and with project owners, fix bugs, enhance features, fork and star file repositories, and submit code.

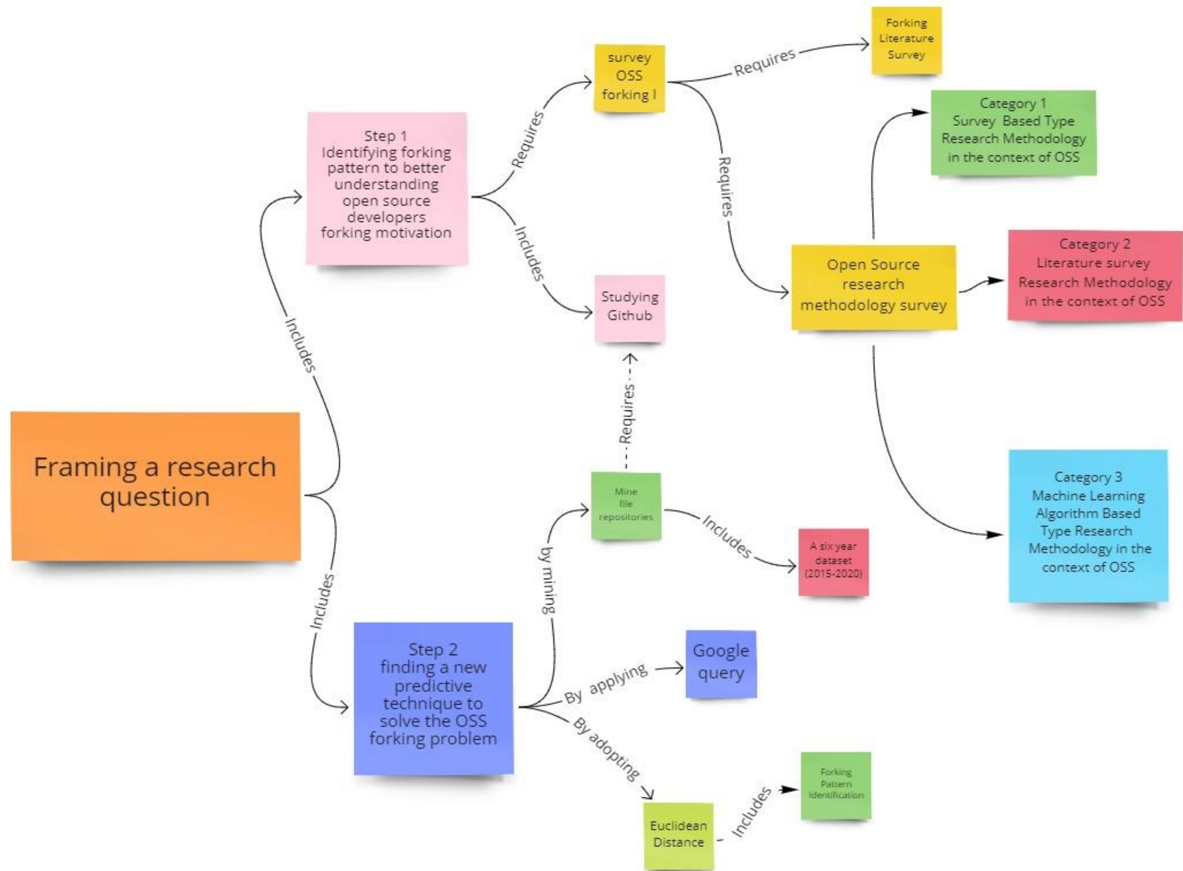

Figure 4. 1: *Literature survey mapping model*

4.3 Chua and Zhang Open Source Software Forking Pattern Prediction Model

Our Chua and Zhang OSS Forking Pattern Prediction Model is shown in Figure 4. 2. We created it during this PhD study to identify and predict OSS forking patterns after analysing monthly fork data performance. It shows three types of developer forking behaviour patterns: 1) fork once only; 2) fork intermittently; and 3) fork steadily. Developer interest and learning experience can be detected from the three forking patterns. The difference between the patterns is that some developers forked once only and never intended to contribute; some forked source codes occasionally that were relevant and of interest to them to fix or provide feedback to project owners for a short period of time; and some developers forked

steadily as they are actively involved in the project throughout or frequently follow projects. To conduct the prediction, we tested our model by mining repositories from a GitHub dataset, analysed the related forking features, then applied the KNN algorithm using the Euclidean distance metric.

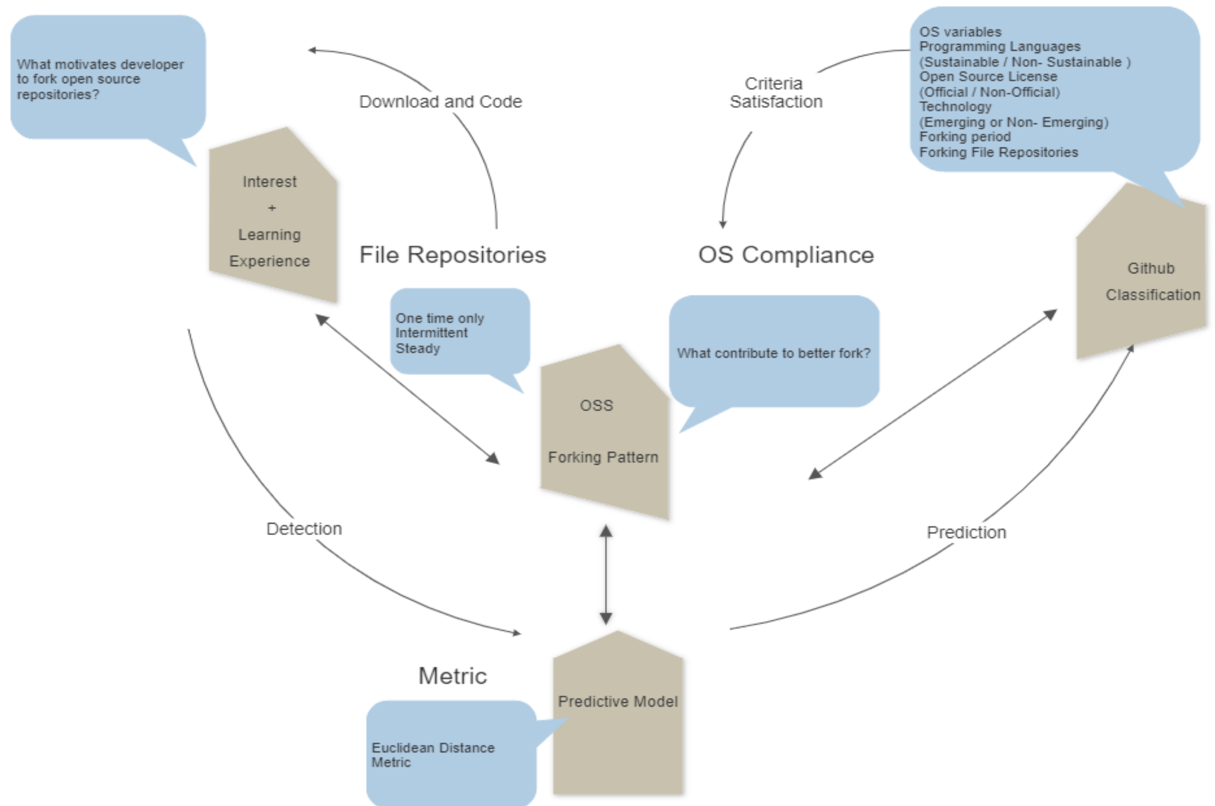

Figure 4. 2: The Chua and Zhang OSS forking pattern prediction model

Chapter 5: A Pilot Study

5.1 Overview

In this chapter, we introduce Euclidean distance to solve the problem of OSS forking performance for programming language repository longevity. This work has been accepted by OpenSym'19 [100].

5.2 Motivation

The motivation of writing the paper for this chapter was to identify forking patterns and predict the distance between repositors longevity and forking data using an Euclidean distance metric to determine fork performance optimisation, based on programming language file repository compliance classifiers, which we analysed for a year-old dataset from GitHub of 47,000 forking instances in 1000 projects.

Despite a vast of literature on programming language popularity and successability, there are very few studies on repositories' programming language survivability in response to forking conditions. As far as we are aware a high number of repository programming languages is not sufficient to ensure good forking performance. To address this issue and assist project owners in adopting the right programming language, it is necessary to predict programming language survivability from forking in repositories. This chapter therefore addresses two related questions:

- are there statistically meaningful patterns within repository data, and, if so,
- can these patterns be used to predict programming language survival?

To answer these questions, we analysed 47,000 forking instances in 1000 GitHub projects. The anecdotal evidence showed long-lived programming languages have a positive impact of extending file repositories and fork longevity whereas short-lived programming languages are less able to drive the file repositories and fork continuity.

5.3 Background

Programming language survivability can be predicted in different ways, with different evaluative methods generating different predictive results. Forking is sometimes ignored when predicting repositories' programming language survivability in GitHub, as the GitHub forking function is an essential mechanism to assist OS developers to quickly code software with support from the internal and external community.

Whether a programming language can survive (perform) or not is highly dependent on forking performance. Each repository file is tied to a programming language that provides developers the freedom to copy and fork the file. Forking features include speed, size and type. Speed refers to the forking period in days, weeks or months; size refers to the number of developers who fork the file; and type refers to source code file characteristics, such as programming language, licence compliance, etc.

However, forking has some challenges; for example, forking performance could be a "high demand but low supply", "low demand and supply" or "low demand but high supply" situation. For example, "high demand but low supply" may reflect using a popular programming language but the repository is not forked by many developers. Conversely, "low demand and supply" may be a niche

programming language, developers and market, e.g., using R language for statistics and data analytics. “Low demand but high supply” may be a new and popular programming language – Swift, Objective C – that developers would be more likely to adopt because of language migration.

It is unclear what causes uncertainty in low or high forking count, affecting programming language survivability. Our research therefore focuses on programming language survivability. This research is critical as an increasing number of repository files are adopted to sustain programming languages, but creators may not be able to find the right developers to fork their language. Further, there is a recent decline in forking as correct programming languages are not being adopted onto repository files. To tackle these issues, our goal in this paper is to report evidence of the effect of forking in programming language repository files.

We are the first researchers to analyse a large forking dataset for developer forking behaviour based on repository file characteristics to predict sustainable programming language survivability. We are also the first to adopt a machine learning method – K Nearest Neighbour (KNN) – to predict sustainable programming language survivability and introduce a robust method to evaluate OS file forking success ability.

5.4 Forking Patterns

Regardless of forking type, there are three forking patterns that can be identified in GitHub: single, or once only; intermittent; and steady. A single fork pattern refers to developers who fork programming language repository files once a month and then not at all in consecutive months. Intermittent refers to forking over

some months, then not in others, then again in later months. A steady fork pattern refers forking files consistently for a defined period, such as every month for 12 months (**Error! Reference source not found.**).

Table 5. 1: Fork patterns

Repository	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Fork Pattern
Rick/ dotfiles	1	0	0	0	0	0	0	0	0	0	0	0	Single
Droogans/ unmaint-ainable code	4	0	4	2	4	0	2	69	3	3	2	2	Interm-ittent
Electron/ electron	287	260	266	198	229	225	191	190	164	223	183	175	Steady

5.5 Software Survival and Programming Language Survival Importance

Many critical factors have been discussed in the literature on success of closed source language development [101] but a focus on programming language assessment must continue in the new OSS development culture. Not only can we expect voluminous source codes to be contributed by developers but also an increase of new OS programming languages added, driving competition for programming language survival.

The survival of a programming language is critical to a repository and a project owner, as a language without forking is equivalent to no new source code, implying no development, potentially as the chosen programming language failed

to produce source code that was ready in time to develop and deliver a software product. In other words, it is difficult to develop a programming language used in a repository file quickly and submit it to a production environment. The longer a repository file remains in GitHub without developer interest, the greater the likelihood of termination once the public repository file expires. It is therefore a waste of development time and effort to create that repository file.

A surviving programming language is one that is more open to interoperability and integration to build ecosystems and emerging technology agility and mobility. A surviving programming language can also reduce the risk of replacing another programming language and developing other components.

Ranking of popular or sustainable programming languages is one way to assure developers a language is reliable to adopt. Unfortunately, however, programming language popularity, sustainability and successability assessments vary across companies, projects, platforms and communities [7, 16], making comparing results difficult.

There is no one method to assess programming language popularity and rank importance regardless of how the language is used and adopted. There is limited literature on assessing programming language popularity, success and sustainability by measuring fork performance as, to date, forking has not been instrumental as a viable process for time to production on repository files.

Since forking forms an integral part of OSS development, ranking importance of programming languages is relevant. A programming language forking rank result could be of benefit when considering and selecting the right programming language to adopt, use and fork in a platform, and obtain the right community support. However, ranking programming language forking success or popularity

is a challenging task, coupled with several factors to consider, including the target platform, elasticity of a programming language, topic of interest, time to production, programming language fork performance, and community support. Most importantly, both forking and programming language are time-independent and assessing them can be daunting as forking fluctuates inconsistently.

5.6 Survivability Prediction on the K Nearest Neighbour Method

The K Nearest Neighbour (KNN) method is often used in the field of data mining and statistics [97-99] because of its implementation simplicity and significant classification performance that produces more accurate results than many other algorithms.

The success of the KNN method lies in its simplicity, ease of use and accuracy [90, 92, 94, 97-99]. In fact, the method is widely used in the field of statistics and data mining due to its implementation simplicity and the fact that its classification performance produces more accurate results than many other algorithms [92-94]. In this paper, the KNN method can handle mixed Euclidean distance.

The KNN method includes the following steps:

1. Load the training data and test dataset.
2. Find K-Nearest Neighbours and assign a value to K.
3. Apply Euclidean distance formula to calculate the distance between the query-instance and all the training samples.
4. Sort and determine the distance nearest neighbours based on the Kth minimum distance.
5. End.

We aimed to predict the lifespan of a programming language through forking using KNN, given forks range from a few months to several months. The algorithm calculates Euclidean distance from the 12 months of the forking period based on the forking pattern categories to evaluate which types of programming language repository file are short-lived or long-lived by the minimum distance. We chose a 12-month period rather than days or weeks as we did not find a significant number of forked changes on repositories over the shorter timeframe. We used the three patterns defined above: single, intermittent, and steady.

According to the Euclidean distance formula [93], the distance between two points in a plane with coordinates (x,y) and (a,b) is given by

$$dist((x,y),(a,b)) = \sqrt{(x-a)^2 + (y-b)^2}$$

and the a, b, c and d variables must be numeric. As such, we converted non-numeric variables from a forking dataset downloaded from GitHub (January–December 2017) into numeric variables (**Error! Reference source not found.**).

We adopted one of the queries from [102] into the Google Big Query using the Select Statement and highlighted the condition to retrieve only created forked repositories:

```
SELECT events.repo.name AS events_repo_name, COUNT(DISTINCT events.actor.id) AS
events_actor_count
FROM (SELECT * FROM TABLE_DATE_RANGE
([githubarchive:day.],TIMESTAMP('2017-01-    01'),TIMESTAMP('2017-12-31')))    AS
events
WHERE events.type = 'ForkEvent'
```

We downloaded 1000 repository files from GitHub and randomly categorised them alphabetically.

To satisfy environment compliance around product, programming language and licence, we referenced Open Source Technology's list of top products developers are interested in [21], the top officially recognised OS licences [20], and IEEE's top programming languages and licences adopted in OS repository files [19], namely Python, C, Java, C, C#, PHP, R, JavaScript, Go and Assembly.

Table 5. 2: *Variables defined for programming language survivability*

Name	Description	Source	Variable	Type[a]	Binary
Events_repo_ name	A repository file name	GitHub	x1	C	N/A
Repo_type	Own creation (from the description of the source code link)	NA	x2	C	N/A
Prog Lang Name	Programming language: 1. Python, 2. C++, 3. Java, 4. C, 5. C#, 6. PHP, 7. R, 8. JavaScript, 9. Go, 10. Assembly	[19]	x3..x13	C	1 Yes, 0 No
OS recognised licence	BSD 3-Clause 'new' or 'revised' licence, BSD 2-Clause 'simplified' or 'FreeBSD' licence, GNU general public licence (GPL), GNU library or 'lesser' general public licence (LGPL), MIT licence, Mozilla, Common development and distribution licence (CCDL), Public licence 2.0, Eclipse public licence	[20]	x14..x21	C	1 Yes, 0 No
OS technology	OpenStack, Progressive Web Apps, Rust, R, cognitive cloud, artificial intelligence, Internet of Things	[21]	x22..x32	C	1 Yes, 0 No
Fork 12	Fork detected every month from Jan to	NA	x33	B	1 Yes,

Name	Description	Source	Variable	Type[a]	Binary
surviving months	Dec				0 No
Environment compliance	Satisfy environment compliance: sustainable top 10 programming language, recognised licence, OS technology	NA	x34	B	1 Yes, 0 No
Forking month	January to December	NA	x35..x47	N	N/A

[a] binary, B; character, C; numeric, N.

A repository file name is a name given to uniquely identify a piece of source code stored in GitHub. Due to some filenames being non-interpretable, the conversion from characters into binary is difficult, e.g., a repository file name labelled "1ppm/1ppmLog". For all variables except the repository file name, attributes with text characters were converted into binary numbers (1=yes, 0=no); e.g., programming language names, repository file and licence. For instance, for the programming language JavaScript, 1 indicated JavaScript was used and 0 indicated it was not JavaScript.

In total, there were 47 attributes, with two added to determine duration of programming language repository file survival (in months) and how many repository files complied with the criteria published in [20, 21].

Our definition of a long-lived programming language repository file was based on detecting a consecutive 12- month forking performance; short-lived was no fork counts detected in the 12-month period. For example, a JavaScript social media repository file was predicted to have a short-lived outcome as there was no fork

in the 12-month period versus a Python machine learning repository file was predicted to be long-lived, having visible monthly forking.

In total, 47,000 forking data over the 12-month period were evaluated for Euclidean distance, using the three patterns, to determine which programming language repository files were short- or long-lived by the minimum distance.

5.7 Programming Language Repository File Categorisation and Fork Pattern Classifiers

Categorising the forking dataset into single, intermittent or steady patterns revealed nine types of programming language repository files based on environment compliance and fork performance (**Error! Reference source not found.**).

Table 5. 3: *Forking patterns*

Forking Pattern	Programming Language Repository Files
Single	Specific repository file (SPF)
Intermittent	Specific repository file met official licence compliance and adopted a modern sustainable programming language (SRFMSPL)
	Specific repository file met official licence compliance (SRFOL)
	Specific repository file met official licence adopted a traditional sustainable programming language (SRFOLTSPL)
	Specific repository file adopted a traditional sustainable programming language (SRFTSPL)
Steady	Specific repository file that did not meet the full environment licence but has healthy fork (SRFHF)
	Specific repository file met official licence compliance that has healthy fork (SRFOLHF)
	Specific repository file met official licence compliance and adopted a

Forking Pattern	Programming Language Repository Files
	modern sustainable programming language that has healthy fork (SRFOLMSPLHF)
	Specific repository file adopted a traditional sustainable programming language that has healthy fork (SRFTSPLHF)

5.8 Classifier Results

The results of using Euclidian distance to categorise the programming language repository file forks are shown in **Error! Reference source not found.** and **Error! Reference source not found.**. A high number were short-lived (79.4%) and only a small number were long-lived (20.6%).

Table 5. 4: Categorising programming language repository files forks as short- or long-lived

Long-lived		Short-lived	
Abbreviation	#	**Abbreviation**	#
SPF	138	SRFOLHF	32
SRFOL	104	SRFHF	20
SRFTSPL	172	SRFTSPLHF	42

Figure 5. 1: Categorising programming language repository file forks as short- or long-lived.

Our results identified that some non-sustainable programming languages that lacked environment compliance survived as long as sustainable programming languages that met environment compliance: 94/206 repository files did not completely meet environment compliance but survived well, e.g., CSS, Kotlin, Emacs Lisp and Jupiter Notebook. Long fork survival could be due to a developer community supporting an OS technology trend, e.g., machine learning, web applications or android operating systems.

We found the majority of sustainable programming languages were short-lived because of low or no licence compliance. The data revealed many developers chose a low compliance licence – development mountain copyright, CC-NC-SA, Creative Commons Attribution 4.1, WTFPL, or Educational Content Licence – however, as these licences are less popular and/or have low compliance some developers are hesitant to contribute [20]. In contrast, long-lived programming language repository files aligned with the top 10 sustainable programming languages [19]. Nevertheless, some repository files that adopted programming

languages not in the top 10 – such as Python, PHP, Swift, Shell and Ruby – also survived well.

5.9 K Nearest Neighbour Results

The coordinates value (of forking data) is defined as how the X-axis refers to the environment compliance and the Y-axis refers to the forking period. The threshold for classification is based on averaging monthly forking data. The ground truth for testing the classification result is to confirm if it is true that high visible fork data can only be only detected on repositories that fully satisfy environmental compliance.

We applied Euclidean distance to calculate each file repository environment compliance distance by comparing the distance of the first repository with the distance of the last repository. We then ranked them according to distance length, with the shortest distance being the lowest the rank is and the longest the being the highest the rank. For instance, under the file classifications, SRFTSPLHF has a Euclidean distance of 1, and a rank of 109.

The results of the KNN method are summarised in **Error! Reference source not found.**, which shows the classification of programming language repository files by KNN/ Euclidean distance. These are illustrated with four case studies below.

Table 5.5: *Categorising programming language repository files sorted by Euclidean distance*

Classification	File Count	Euclidean Distance	Rank
SRFOTLSPLHF/SRFOLMSPLHF	113	0	1
SRFTSPLHF	41	1	109
SRFOLHF/SRFOTLSPLHF	33	1	113

Classification	File Count	Euclidean Distance	Rank
SRFOLSPL/SRFOLMSPL	374	1.4	187
SRFHF	20	2	562
SRF	1	3.2	566
SRFSPL/SRFOL	281	2.2	582
SRF	137	3.2	863

5.9.1 Case One

A programming language repository file is found to associate with the following properties: one of the top ten OS technologies [21], met legitimate licence compliance [20], adopted a sustainable programming language [19], and displayed monthly forking over the last 12 months. This file is predicted to be a long-lived surviving programming language file with healthy forking. Our results predict SRFOTLSPLHF or SRFMTLSPLHF would fall under this category.

5.9.2 Case Two

A programming language repository file is found not to associate with one of the following properties: one of the top ten OS technologies [21], met legitimate licence compliance [20], adopted a sustainable programming language [19], and displayed monthly forking over the last 12 months. This file is predicted to be a long-lived surviving programming language file with healthy forking. Our results predict SRFHF would fall under this category.

5.9.3 Case Three

A programming language repository file is found not to associate with more than one of the following properties: one of the top ten OS technologies [21], met

legitimate licence compliance [20], adopted a sustainable programming language [19], and did not display monthly forking over the last 12 months. This file is predicted to be a lower surviving programming language. Our results predict SRF would fall under this category.

5.9.4 Case Four

A programming language repository file is found not to associate with more than one of the following properties: one of the top ten OS technologies [21], met legitimate licence compliance [20], adopted a sustainable programming language [19], but displayed monthly forking over the last 12 months. This file is predicted to be a lower surviving programming language. Our results predict SRFOL would fall under this category.

5.10 Evaluation

In this paper, we proposed evaluating sensitivity and specificity to describe test performance, as these parameters remain true regardless of the population of programming language repository files to which the test is applied.

Definitions of environment compliance parameters are presented in **Error! Reference source not found.**, where: true positive (TP) is defined as the number of programming language repository files that met environment compliance and were classified as long-lived; false positive (FP) is defines as the number that met environment compliance and were mistakenly classified as short-lived; true negative (TN) is defined as the number that did not meet environment compliance and were classified as long-lived; and false negative (FN) is defined as the number

that did not meet environment compliance and were mistakenly classified as short-lived.

Table 5. 6: *Environment compliance*

Full Compliance	Long-lived	Short-lived	Total
Yes	111 (TP)	94 (FP)	205
No	0 (FN)	795 (TN)	795
	111	889	1000

For this study, we classified the data into training (80%) and testing (20%) samples. We used the KNN method to classify the class of the repository files then calculated the Euclidean distance between the forking period and forking pattern. After determining the parameter *k* and running the KNN algorithm, accuracy was calculated using precision, sensitivity and specificity. The formulas of the four measures [93] are outlined below.

Accuracy refers to the proportion of true results from the number of programming language repository files that met environment compliance and the true negative results from the number that did not meet environment compliance and were classified as long-lived

$$\text{Accuracy} = \text{TP+TN/(TP+TN+FP+FN)}$$

For this study, accuracy is 111+795/(111+795+94+0)= 0.906 (90.6%).

Precision refers the ratio of correctly predicted all programming repository files that appeared to survive, how many have actually survived? High precision therefore relates to a low false positive rate.

$$\text{Precision} = \text{TP/TP+FP}$$

For this study, precision is 111/111+94=0.542 (54.2%). **Error! Reference source not found.** summarises all four metrics.

Sensitivity refers the proportion of long-lived programming language repository files that meet full environment compliance, and **specificity** refers the proportion of short-lived programming language repository files that meet full environment compliance. Hence, the formula is

$$\text{Sensitivity} = \text{TP}/\text{TP}+\text{FN} \qquad \text{Specificity} = \text{TN}/\text{TN}+\text{FP}$$

For this study, sensitivity is 111/111+0=1 (1%) and specificity is 795/795+94=0.894 (89.4%).

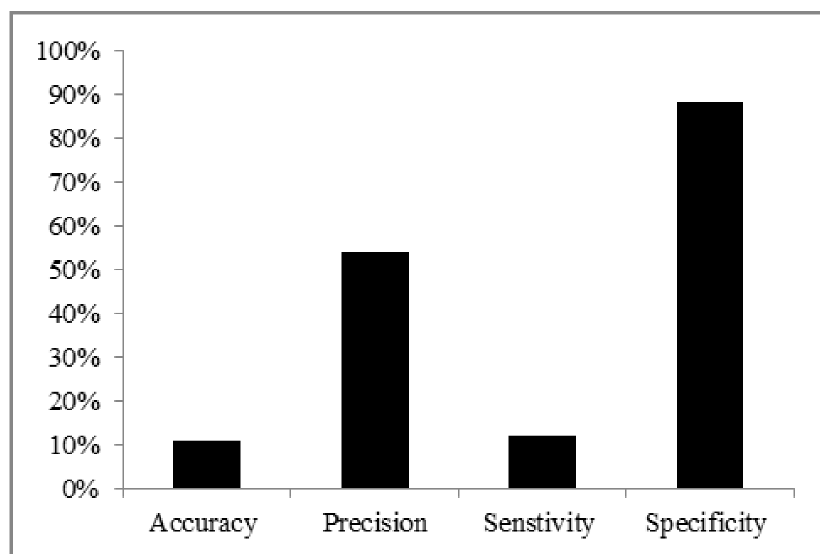

Figure 5. 2: *Evaluative results comparison of the dataset*

5.11 Conclusions and Future Work

Error! Reference source not found. highlights that there are less long-lived programming language repository files than short-lived. For a programming language repository file to survive it must satisfy environment compliance properties; the data reveal most do not comply and are therefore short-lived. In

other words, many project developers or owners who created repository files may have ignored, or failed to pay attention to, environment compliance factors, such as technology trends and licensing.

Error! Reference source not found.6 is a statistical overview of programming language repository file lifespan and **Error! Reference source not found.** is an overview of the test result accuracy showing a breakdown of accuracy, precision, sensitivity and specificity. Our findings reveal that it is necessary for developers to pay attention to environment compliance before developing a repository file if they want to ensure healthy forking and file survivability.

The predictive results help us to better categorise developers' motivations for forking. The existing literature identified seven categories of forking: OS, project, software, social, code, programming language and repository. Our data show long-lived forked programming language repositories that satisfy environment compliance are potentially related to social, programming language and repository forking. In contrast, short-lived forked programming languages that are environment compliant are related to code, OS and project forking.

Our future work in this area will focus on introducing new environment compliance variables to fast-growing project code that is forked from very large-scale programming languages with boundary conditions. In addition, we will evaluate which machine learning method can accurately and reliably predict fork patterns for short-lived and long-lived programming languages.

Chapter 6: A Longitudinal Study

6.1 Overview

This chapter presents a paper that extends the work in Chapter 5. The same research method was used except this time we studied and analysed the dataset over a six-year timeframe instead of one year. The research paper is under review.

The importance of this study is the evidence gathered on recognising the role of repository compliance as an attribute in forking sustainability. A complete compliance programming language repository can attract developers to fork and maintain fork sustainability, whereas programming language repositories with partial or no compliance are less likely to maintain their fork sustainability.

This chapter also introduces data normalisation, which was applied due to the large fluctuation in monthly fork count – from tens to thousands – which made it difficult to predict distance.

6. 2 Motivation

The motivation of this chapter was to conduct a longitudinal study that used longitudinal forking data over a six-year period validated under the same KNN Euclidian distance method. The study objective was to observe and collect project data on a number of variables without trying to interrupt or influence variables. In addition, we wanted to examine the same repository to detect any changes of forking data that might occur over a longer period of time.

6.3 Background

According to Jiang et al. [7], there are three main reasons why developers want to fork. 1) They want to modify and improve source codes by fixing bugs, adding new features and making copies in GitHub. 2) They want to learn a programming language, so choose files to fork in their preferred language. Or, 3) The repositories are attractive. Other less common motivations may be to create a new project or repository in response to team conflicts or to find a job as a coder.

Chua et al. [27] recently conducted a Systematic Literature Review (SLR) that identified reasons for, and challenges associated with, OS forking [26]. They identified twenty-three factors across three categories. Firstly, *forking motivation*, which includes coding for: revising requirements [40], job seeking [42], licensing compliance [7, 36, 42] or software compliance [1, 9, 16, 46]; coding to extend the duration of an original project development [16, 46, 47] or community social coding development [36]; coding to address ownership implications [8, 44, 46, 47] or business strategy risks [36], or risks associated with team coding skill inequality [3], divergent specialisation [3, 46, 47], misaligned objectives [3, 46, 47], poor leadership [5, 46, 47] or cultural differences [3, 5, 46]; coding for community socialisation [5, 34, 36] or by socialising [34, 95], or for software activity [16, 47] or the ecosystem [16, 47].

The second category is *forking sustainability* with the primary factor of community activity [7, 40, 44]. The third category is *forking lessons learnt* with factors including presence of a formal process [43], legal implications [15, 18], transfership [6], product expertise shortage [45] and upgrading a developer role to

a product role [6, 15, 45]. All these factors are reasons to justify why developers want to fork.

There is limited research on how forking motivation aligns with aspects of OS infrastructure support, such as combined OS licence compliance, programming language sustainability and OS technology. Our study focused on deepening understanding of developers' forking motivation as our philosophical view on healthy fork performance is based on the alignment of developer forking motivation and OS infrastructure support. A large dataset of monthly forking from 2015 to 2020 with 2–4-digit fork counts was downloaded from our previous study [48] for analysis, classification and prediction.

We are the first group of researchers to conduct a comprehensive analysis of healthy file fork repository (HFFR) to normalise data before applying the KNN machine learning method to predict HFFR classifications.

6.4 Fork Pattern Identification and Data Collection

In our previous work [100], 47,000 forking instances in 1000 GitHub projects were downloaded and analysed. We identified three forking patterns: 1) single (once only), 2) intermittent, and 3) steady. A single fork pattern refers to developers who fork programming language repository files once a month and then not at all in consecutive months. Intermittent refers to forking over some months, then not in others, then again in later months. A steady fork pattern refers to forking files consistently for a defined period, such as every month for 12 months (**Error! Reference source not found.**).

Table 6. 1: Examples of file repository monthly forking

Repository	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Fork Pattern
adam-p/ markdown-here	1	0	0	0	0	0	0	0	0	0	0	0	Single
Airbnb/ JavaScript	4	0	4	2	4	0	2	69	3	3	2	2	Interm-ittent
Alamofire/ Alamofire	287	260	266	198	229	225	193	190	164	223	183	175	Steady

In this paper, our aim was to investigate HFFR performance. This study extended our previous work [100]; we adopted one of the queries from [100] into a Google Big Query using a select statement that highlighted the condition to retrieve only created forked repositories (**Error! Reference source not found.**).

Table 6. 2: Big query statement

Select statement for Google Big Query

```
SELECT events.repo.name AS events_repo_name,

COUNT(DISTINCT events.actor.id) AS events_actor_count

FROM (SELECT * FROM TABLE_DATE_RANGE ([githubarchive:day.],

TIMESTAMP('2017-01- 01'),TIMESTAMP('2017-12-31'))) AS events

WHERE events.type = 'ForkEvent'
```

Of the 1000 file repositories from [100] retrieved over a period of 6 years (72 months), 62 met the study criterion of being a healthy fork repository, defined as

having been forked by developers every month from Jan 2015 to Dec 2020 (**Error! Reference source not found.**).

Table 6. 3: Forking data of selected file repositories, 2015–2020

Repository Name	Year												Fork Status
	J	F	M	A	M	J	J	A	S	O	N	D	
2015													
numpy/ numpy	53	57	95	56	45	52	55	48	51	63	73	86	Steady
Airbnb/ JavaScript	135	132	346	490	199	221	514	265	261	240	248	274	Steady
nightscout/ cgm-remote-monitor	408	278	839	208	199	155	168	205	251	198	169	224	Steady
2016													
numpy/ numpy	57	69	69	55	59	58	61	60	61	75	56	53	Steady
Airbnb/ JavaScript	319	312	425	375	326	325	341	384	338	357	361	323	Steady
nightscout/ cgm-remote-monitor	296	259	323	296	364	374	366	330	432	509	442	426	Steady
2017													
numpy/ numpy	69	80	78	71	80	77	66	73	72	91	104	91	Steady
Airbnb/ JavaScript	355	410	447	345	371	378	432	384	348	331	440	328	Steady
nightscout/ cgm-remote-monitor	446	440	392	420	444	505 6	412	430	578	580	545	663	Steady
2018													
numpy/ numpy	99	87	75	81	70	80	91	75	85	85	94	79	Steady
Airbnb/ JavaScript	362	289	346	306	305	267	303	297	248	285	270	221	Steady
nightscout/ cgm-remote-monitor	788	828	823	696	739	719	698	823	863	788	868	884	Steady

Repository Name	Year												Fork Status
	J	F	M	A	M	J	J	A	S	O	N	D	
2019													
numpy/ numpy	97	90	116	121	110	99	125	106	106	130	113	100	Steady
Airbnb/ JavaScript	259	306	294	308	293	263	284	238	238	260	229	208	Steady
nightscout/ cgm-remote-monitor	1028	1086	1172	1184	1260	1154	1373	1177	1177	1287	1327	1378	Steady
2020													
numpy/ numpy	134	111	120	135	161	130	141	93	132	116	116	116	Steady
Airbnb/ JavaScript	192	283	260	275	332	296	259	225	226	279	214	293	Steady
nightscout/ cgm-remote-monitor	1528	1246	931	712	912	954	1086	1298	1800	4015	4015	1730	Steady

6.5 Normalisation and Euclidean Distance

For this study, we divided the data into training (80%) and testing (20%) samples. The 108 variables used to define a HFFR are shown in **Error! Reference source not found.**.

Table 6. 4: Variables defined for a healthy fork file repository

Name	**Description**	**Source**	**Variable**	**Type**	**Binary**
Event_repo_ name	Repository file name	GitHub	x1	C	N/A
Repo_type	Own creation (from source code link description)	NA	x2	C	N/A
Programming language name	1, Python; 2, C++; 3, Java; 4, C; 5, C#; 6, PHP; 7, R; 8, JavaScript; 9, Go; 10, Assembly	[100]	x3..x13	C	1 Yes, 0 No
OS licence	BSD 3-Clause 'new' or 'revised' licence; BSD 2-Clause 'simplified' or 'FreeBSD' licence; GNU general public licence (GPL); GNU library or 'lesser' general public licence (LGPL); MIT licence; Mozilla; Common development and distribution licence (CCDL); Public licence 2.0; Eclipse public licence	[20]	x14..x21	C	1 Yes, 0 No
OS technology	OpenStack; Progressive web apps; Rust; R; cognitive cloud; artificial intelligence (AI); Internet of Things	[21]	x22..x32	C	1 Yes, 0 No
Fork 72 surviving months	Fork detected every month from Jan 2015 to Dec 2020	N/A	x33	B	1 Yes, 0 No
Environment compliance	Satisfy environment compliance: Sustainable top 10 programming language, recognised licence, OS technology	N/A	x34	B	1 Yes, 0 No
Forking month	January 2015 to December 2020	N/A	x35..x107	N	NA
HFFR type	Each repository file contains a fork count across the 72 months	N/A	x108	C	SRFHF, SRFOLHF, SRFOLMSPLHF, SRFTSPLHF, SRFOLTSPLHF

*Binary, B; Character, C; Numeric, N. N/A=Not applicable.

The large fluctuation in monthly fork count – from tens to thousands – makes it difficult to predict distance so the fork count values first needed to be normalised to reduce this range. For instance, one variable may be binary while another may be a number with two, three or four digits. Normalising enables comparisons and meaningful correlations, and can be done using the following method [103]:

$$Z = (x + \mu)/\sigma$$

Z = normalisation result

x = mean of the sample

μ = mean of the population

σ = standard deviation of the population

Error! Reference source not found. presents the same forking data after normalisation.

Table 6. 5: Forking in 5 years (2015-2020) after normalisation

Repository Name	Year												Fork Status
	J	F	M	A	M	J	J	A	S	O	N	D	
2015													
numpy/ numpy	-0.76	-0.74	-0.16	-0.07	-0.61	-0.57	-0.04	-0.42	-0.44	-0.42	-0.38	-0.29	Steady
Airbnb/ JavaScript	-0.90	-0.88	-0.76	-0.88	-0.90	-0.89	-0.88	-0.88	-0.88	-0.84	-0.81	-0.77	Steady
nightscout/ cgm-remote-monitor	-0.29	-0.47	1.02	-0.60	-0.61	-0.70	-0.68	-0.55	-0.46	-0.52	-0.58	-0.42	Steady
2016													
numpy/ numpy	-0.27	-0.21	0.02	-0.01	-0.16	-0.09	-0.07	0.06	-0.07	-0.08	0.04	0.03	Steady
Airbnb/ JavaScript	-0.86	-0.82	-0.83	-0.85	-0.84	-0.83	-0.83	-0.83	-0.82	-0.80	-0.83	-0.82	Steady
nightscout/ cgm-	-0.32	-0.34	-0.23	-0.22	-0.07	0.05	0.00	-0.09	0.19	0.31	0.27	0.35	Steady

Repository Name	Year												Fork Status
	J	F	M	A	M	J	J	A	S	O	N	D	
remote-monitor													
2017													
numpy/ numpy	0.03	0.18	0.07	-0.07	0.00	-0.13	0.14	0.00	-0.27	-0.15	0.15	-0.12	Steady
Airbnb/ JavaScript	-0.79	-0.76	-0.81	-0.80	-0.78	-0.81	-0.82	-0.80	-0.84	-0.76	-0.72	-0.75	Steady
nightscout/ cgm-remote-monitor	0.29	0.26	-0.06	0.13	0.20	10.53	0.09	0.12	0.20	0.48	0.43	0.78	Steady
2018													
numpy/ numpy	-0.13	-0.16	-0.80	-0.17	-0.21	-0.17	-0.06	-0.09	-0.36	-0.15	-0.13	-0.22	Steady
Airbnb/ JavaScript	-0.76	-0.74	-0.95	-0.78	-0.81	-0.75	-0.71	-0.77	-0.78	-0.74	-0.69	-0.72	Steady
nightscout/ cgm-remote-monitor	0.88	1.38	-0.55	0.87	0.90	1.24	1.16	1.50	1.21	1.35	1.77	2.12	Steady
2019													
numpy/ numpy	-0.44	-0.06	-0.26	-0.37	-0.25	-0.25	-0.25	-0.32	-0.32	-0.29	-0.32	-0.36	Steady
Airbnb/ JavaScript	-0.79	-0.72	-0.71	-0.75	-0.71	-0.71	-0.67	-0.69	-0.69	-0.64	-0.66	-0.69	Steady
nightscout/ cgm-remote-monitor	1.19	2.31	1.91	1.40	2.22	2.28	2.59	2.34	2.34	2.49	2.88	3.17	Steady
2020													
numpy/ numpy	-0.42	-0.14	-0.32	-0.41	-0.34	-0.36	-0.44	-0.43	-0.45	-0.31	-0.46	-0.17	Steady
Airbnb/ JavaScript	-0.60	-0.66	-0.68	-0.71	-0.68	-0.71	-0.69	-0.76	-0.67	-0.71	-0.70	-0.67	Steady
nightscout/ cgm-remote-monitor	3.55	2.76	1.42	0.52	0.79	1.06	1.34	2.22	3.34	8.86	8.97	3.85	Steady

After normalisation, we then applied Euclidean distance [90] to calculate the distance between HFFR types and the OS infrastructure support.

The equation [90] is as follows:

$$d(p,q) = \sqrt{\left(\sum_{i=1}^{n}(q_i - p_i)\right)^2}$$

p, q = two points in Euclidean n-space

q_i, p_i = Euclidean vectors, starting from the origin of the space (initial point)

n = n-space

The algorithm calculates Euclidean distance over the 72 months of the forking period based on pattern categories to evaluate the classifier HFFR accuracy. We chose a 72-month period to identify long-lived HFFRs. **Error! Reference source not found.** outlines the total counts for the five HFFRs.

Table 6. 6: Healthy fork file repository types and counts

No.	OS Infra-structure Compliance Cluster	Healthy Fork File Repository Type		Total count
		Description	Abbreviation	
1	None	Did not meet environment licence	SRFHF	6
2	Partial	Met official licence compliance	SRFOLHF	13
3	Partial	Adopted traditional sustainable programming language	SRFTSPLHF	3
4	Full	Met official licence compliance Adopted modern sustainable	SRFOLMSPLHF	2

No.	OS Infra-structure Compliance Cluster	Healthy Fork File Repository Type		Total count
		Description	Abbreviation	
		programming language		
5	Full	Met official licence compliance Adopted traditional sustainable programming language	SRFOLTSPLHF	38

6.6 Results

We used the KNN method to classify HFFRs then calculated the Euclidean distance between the forking period and forking pattern. **Error! Reference source not found.** shows the clusters with full compliance – SRFOLTSPLHF and SRFOLMSPLHF – have Euclidean distances of 4 and 4.6. For the cluster groups with partial compliance, SRFOLHF had non-sustainable programming languages whereas SRFTSPLHF did not comply, using Creative Commons Attribution Non-Commercial ShareAlike (CC-NC-SA) or undeclared licences. The Euclidian distances were 9.1 for SRFOLHF and 8.5 for SRFTSPLHF, with a difference of 0.6.

Table 6. 7: Healthy fork file repository types ranked by Euclidean distance

OS Infrastructure Compliance Cluster	Classification	Rank	Euclidean Distance
Full	SRFOLTSPLHF	1	4
Full	SRFOLMSPLHF	13	4.6
Partial	SRFOLHF	18	8.5
Partial	SRFTSPLHF	19	9.1

None	SRFHF	33	17.5

The remaining cluster group was SRFHF, which was non-compliant with an Euclidean distance of 17.5, the furthest away from the other two clusters. These distances draw our perspective on a deeper understanding of developer forking motivation, by showing that compliance is positively associated with motivation, from multiple environments in multiple disciplines. For example, moodle/moodle is an HFFR forked by developers who want to know how to build e-learning platforms. These developers do not work individually but are in software development companies or e-learning environments. For example, the "rdpeng/ExData_Plotting1" file repository is forked heavily by students, researchers, and statisticians for data analytics.

Logically, a HFFR is a repository that should have the most compliance. However, our findings show that some HFFRs have partial or no compliance. We therefore tested K based on 1, 3, 5, 10, 13, 14, 15, 18, 19, 20, 30, 40, 50 and 60 (**Error! Reference source not found.**; **Error! Reference source not found.**). **Error! Reference source not found.** shows the results for K = 1, 3, 5, 10, 15, 20 using SRFOLTSPLHF as the predictive file. In this dataset, the majority of the file repositories comply to with OS infrastructure licences and adopt a sustainable programming language. There is a small percentage of partial and non-compliance HFFRs detected in the dataset – SRFOLHF, SFTSPLHF and SRFHF – when K is ranked 18, 19, 50 and 60.

Table 6. 8: Healthy fork file repository types ranked by Euclidean distance

K	HFFR	K	HFFR
1	SRFOLTSPLHF	18	SRFOLHF

3	SRFOLTSPLHF	19	SRFTSPLHF
5	SRFOLTSPLHF	20	SRFOLTSPLHF
10	SRFOLTSPLHF	30	SRFOLTSPLHF
13	SRFOLMSPLHF	40	SRFOLTSPLHF
14	SRFOLMSPLHF	50	SRFHF
15	SRFOLTSPLHF	60	SRFHF

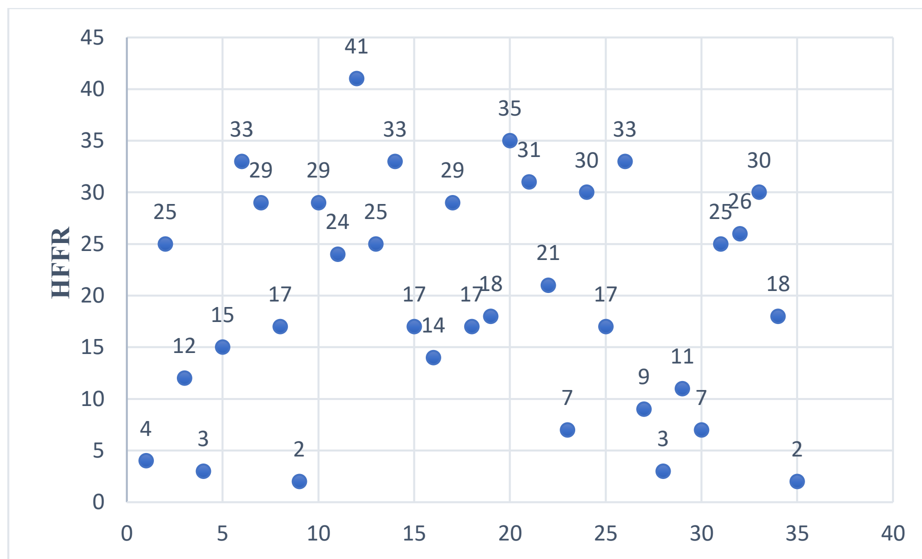

Figure 6. 1: Euclidean distance ranking

6.7 Evaluative Test Results

To evaluate test performance, we evaluated accuracy, precision, sensitivity, and specificity, as these parameters remain true regardless of the population of HFFRs to which the test is applied. Definitions of OS compliance parameters are presented in **Error! Reference source not found.** in relation to identifying fork in a population through a diagnostic test. In this study, true positive (TP) refers to the number of HFFRs that met environment compliance and were classified as healthy forking; a false positive (FP) refers to the number of HFFRs that did not

meet environment compliance but were mistakenly classified as healthy forking; true negative (TN) refers the number of HFFRS that did not meet environment compliance and were classified as healthy forking; and false negative (FN) refers to the number of HFFRS that did not meet environment compliance but were mistakenly classified as healthy forking.

Accuracy refers to the proportion of true results of HRRFS among the total number of positive and negative cases examined. Precision refers to the ratio of correctly predicted positive observations of HFFRS to the total predicted positive observations; that is, of all HFFRs that appeared to survive, how many survived? High precision therefore relates to a low false positive rate. Sensitivity is the proportion of HFFRs that meet full environment compliance, and specificity is the proportion of HFFRs that do not meet full environment compliance [93].

Table 6. 9: Definition and formula for accuracy, precision, sensitivity and specificity

Algorithm	Example	Formula
Accuracy	Proximity of measurement results to true value	$(\text{True Positive} + \text{True Negative})/(\text{True Positive} + \text{False Positive} + \text{True Negative} + \text{False Negative})$
Precision	Repeatability and reproducibility of measurement	$(\text{True Positive})/(\text{True Positive} + \text{False Positive})$
Sensitivity	Proportion of disease correctly identified	$(\text{True Positive})/(\text{True Positive} + \text{False Negative})$
Specificity	Proportion of healthy patients in who disease	$(\text{True Negative})/(\text{True Positive} + \text{False Negative})$

	correctly excluded	

The four metrics [94] for this study are therefore as follows (also shown in **Error! Reference source not found.**):

Accuracy = (40+6)/(40+6+13+3) = 0.72 or 72%

Precision = 40/40+13=0.75 or 75%.

Sensitivity = 40/(40+3) = 0.93 or 93%

Specificity = 6/13 =0.32 or 32%

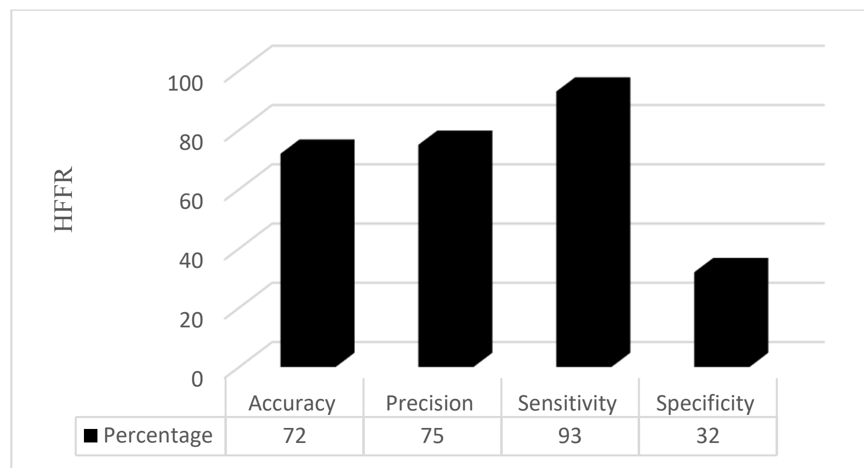

Figure 6. 2: Evaluative results

6.8 Discussion

Our results indicate the majority of healthy forking longevity in OS projects comply with OS licences, which means a well-protected digitally forking environment can make developers realise and trust that particular HFFR is safe to fork. However, this does not equate to being invulnerable to risk of legal copyright

implications. A study of most popular programming languages by country ranked Java second after the world's most popular programming language, C [104]. Fifty-three countries favoured five programming languages – Java, C++, Python, SQL, and Ruby – with Java and C++ most popular [104]. In other words, HFFRs written in any of these five programming languages are more likely to be forked and downloaded. We validated our findings again these studies and confirmed the categories of SRFOLTSPLHF and SRFOLMSPLHF HFFRs have at least one of these top five programming languages. Additionally, SRFOLMSPLHF HFFRs have at least one ancillary programming language, e.g., Swift plus Ruby or Go plus C.

Our results also show a small number of surviving HFFRs do not comply with OS licences and programming language adoption. These HFFRs are not seeking developers to innovate the source code; rather they are forked by developers for downloading purposes, to use for a specific reason, not related to developing a system or an application. That is, forking by users, not developers.

A limitation of our study is that while our test performs well in terms of sensitivity, correctly classifying 93% of HFFRs, it had lower specificity, i.e., correctly excluding only 32% of unhealthy file fork repositories. The test was moderately accurate, with 72% true results, and precise, with 75% of identified HFFRs being 40 healthy.

In conclusion, we predict three types of HFFR clusters and have proven the importance of OS licence compliance and that adopting a suitable and sustainable programming language can motivate developers to fork. We suspect the third group of HFFRs, SRFHF, is highly vulnerable to an OS security threat [105], such as openness and the lack of compliance.

Our future work includes applying a deep learning technique to analyse grouping of HFFRs based on clusters by specific OS licence compliance or adoption of a specific sustainable programming language for its significance to further develop our understanding of developers' forking trust and motivation.

Chapter 7: Conclusion

7.1 Overview

This chapter summarises our conclusions and contributions from the study. We outline salient recommendations from these conclusions and suggest further work that will build on this PhD.

The excitement surrounding forking research has grown in recent years, with an increase in studies of the variables involved in OS forking, including role, activities, type and performance. Particularly, within OS communities, users or developers would like to determine how best to optimise fork performance and project owners want to boost their coding confidence by understanding which methods are most viable and sustainable, or that can be used to predict or monitor their repository fork status – no, low or high forking. Programming language developers are also interested in forking, to learn a new skill, receive an incentive, or find a job or get a promotion.

OSS environment compliance and compatibility is also important in making OSS communities feel safe, secure, and accurate while forking a repository. It can promote forking and help reduce fork code waste. It also offers developers flexible opportunities to download and fork healthier and faster, with no concerns over intellectual property or copyright infringement issues on OS licences. Sustainable programming languages, on the other hand, provide greater coding opportunities for developers to fork and increase the forking chances on other file repositories that are coded in similar programming languages. As such, it translates directly to better forking performance, which in turn increases project performance. Forking

repositories in relation to a new or emerging technology motivates developers likely to fork and increases fork visibility

The research aimed to identify a reliable forking prediction method to solve forking scarcity problem. The results presented in Chapter 5 using Euclidean distance showed that a year-old dataset of programming language file repositories that satisfied OS infrastructure compliance can predict high fork visibility. We applied the same research method to a longitudinal dataset (six years) and also predicted high fork visibility. The empirical data from these studies was discussed in Chapter 6.

Based on our quantitative and qualitative analysis of forking patterns in response to OS development environment compliance we concluded that full environment compliance of a file repository – that is, a sustainable programming language, a legitimate OS licence and a new or emerging technology – can strengthen developers positive forking motivation behaviour, irrespective of time period (one year versus six years).

7.2 Contributions

Our research contributions were as follows:

1. **Interpretation clarity**: We addressed the forking interpretation issue by clarifying forking from a new perspective based on developer forking motivation behaviour. Previous research had focused on identifying reasons for forking, such as personal, project, communities, social network, bugs fixed, etc., In contrast, we focused on an OS infrastructure environmental compliance perspective from a large fork population and concluded the strengthening effect on developer forking motivation.

2. **Forking scarcity**: We addressed the forking scarcity issue by predicting high forking visibility from five file repository classifiers and determined repositories that are less compliant with programming languages, OS licence and technology trends will not generate high fork visibility.
3. **Highly desirable OS variables analysis:** The existing literature, reviewed in Chapters 2 and 3, examined one desirable environment OS variable with respect to developers' forking popularity and successability. We instead reviewed three highly desirable environmental variables: programming languages, OS licence and technology trends to predict low to high forking visibility.
4. **OS forking pattern**: Previous analyses of forking features covered forking size, type and volume. Our analysis was more comprehensive. We analysed monthly forking data and predicted three types of develop fork patterns: single (once only), intermittent (some months with fork counts and some months without) and steady (fork every month).
5. **Euclidean distance, KNN:** Our research method – the Euclidean distance of KNN – showed high accuracy based on the two empirical works evaluated. The prediction accuracy of the model is more precise than other techniques like data mining, regression analysis or descriptive statistics.

7.3 Recommendations

While previous forking prediction methods limited generalisability of results, our approach provides new insight into forking survivability performance. This research clearly illustrates forking prediction method accuracy but it also raised a question on fork survivability analysis. Our dataset does not include other OSS

variables concerned with forking. For instance, activities on fixed bug, feature enhancements, star ranking of a repository, developer demographics, or fork practices, for instance, the intent of forking and multiple fork times as duplicated copies.

Tracking these kinds of activities is time intensive. Moreover, the monthly fork counts we downloaded from GitHub are no longer original once modified or massaged during the Euclidian distance KNN method. The interpretation of these results can therefore be misleading. Our recommendation is to download forking data from the hosting platform directly as data and information are real-time and objective.

7.4 Future Work

Our results are based on predicting high fork visibility from a single machine learning method, Euclidean distance KNN. Further research could validate the impact, and compare accuracy, of other predictive machine learning methods, such as Linear Regression, Decision Tree, Random Forest, Naïve Bayes and Support Vector Machine. The popularity of forking will continue to increase. As such, there is a continued need to explore and extend repository classifications, for example, to classify a homogeneous technology group, a programming language or OS licence clusters for deep learning.

Bibliography

[1] L. A. Meyerovich and A. Rabkin, "Empirical analysis of programming language adoption," presented at the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, Indianapolis, USA, 2013.

[2] "Github." https://Github.com (accessed 5 October 2020).

[3] L. Nyman and T. Mikkonen, "To fork or not to fork: fork motivations in SourceForge projects," *International Journal of Open Source Software & Processes,* vol. 3, no. 3, pp. 1–9, 2011.

[4] J. Freeman. (2015) Bitbucket vs. GitHub: Which project host has the most? *JavaWorld*.

[5] G. Robles and M. Gonzalez-Barahona, "A comprehensive study of software forks: dates, reasons and outcomes," presented at the IFIP International Conference on Open Source Systems, Hammamet, Tunisia, 2012.

[6] F. Ikuine and H. Fujita, "How to avoid fork: The guardians of Denshin 8 Go, Japan," *Annals of Business Administrative Science,* vol. 13, no. 5, pp. 283–298, 2014.

[7] J. Jiang, D. Lo, J. H. He, X. Xia, P. S. Kochlar, and L. Zhang, "Why and how developers fork what from whom in GitHub," *Journal of Empirical Software Engineering Volume,* vol. 22, no. 1, pp. 547–578, 2017.

[8] J. Gamalielesson and B. Lundell, "Sustainability of open source software communities beyond a fork: How and why has the LibreOffice project evolved?," *Journal of Systems and Software,* vol. 89, pp. 128–145, 2013.

[9] G. von Krogh, S. Haefliger, S. Spaeth, and M. W. Wallin, "Carrots and rainbows: Motivation and social practice in open source software development," *Journal of MIS Quarterly,* vol. 36, no. 2, pp. 649–676, 2012, doi: 10.2307/41703471.

[10] G. Hertel, S. Niedner, and S. Herrmann, "Motivation of software developers in Open Source projects: An Internet-based survey of contributors to the Linux kernel," *Journal of Research Policy,* vol. 32, no. 7, pp. 1159–1177, 2003.

[11] S. Balali, I. Steinmacher, U. Annamalai, A. Sarma, and M. Gerosa, "Newcomers' barriers – is that all? An analysis of mentors' and newcomers' barriers in OSS projects," *Computer Supported Cooperative Work,* vol. 27, no. 3, pp. 679–714, 2018.

[12] C. M. Schweik, "Sustainability in open source software commons: Lessons learned from an empirical study of SourceForge projects," *Technology Innovation Management Review,* pp. 13-19, 2013.

[13] A. E. Azarbakht and C. Jensen, "Longitudinal analysis of the run-up to a decision to break-up (fork) in a community," presented at the IFIP International Conference on Open Source Systems: Towards Robust Practices (OSS 2017), 2017.

[14] V. Cosentino, J. L. C. Izquierdo, and J. Cabot, "A systematic mapping study of software development with GitHub," *IEEE Access,* vol. 5, pp. 7173–7192, 2017, doi: 10.1109/ACCESS.2017.2682323.

[15] R. L. Glass, "A sociopolitical look at open source," *Journal of Communications of the ACM,* vol. 46, no. 11, pp. 21–23, 2003.

[16] F. Tegawendé, T. Bissyandé, F. Thung, D. Lo, L. X. Jiang, and L. Réveillère, "Popularity, interoperability and impact of programming languages in 100,000 open source projects," presented at the IEEE 37th Annual Conference Computer Software and Applications Conference (COMPSAC), Kyoto, Japan, 2013.

[17] B. Ray, D. Posnett, V. Filkov, and P. Devanbu, "A large scale study of programming languages and code quality in GitHub," in *22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Hong Kong, China, 2014.

[18] R. Moen, "Fear of Forking," no. 22 November 2016. [Online]. Available: http://linuxmafia.com/faq/Licensing_and_Law/forking.html

[19] IEEE Spectrum. "The Top Programming Languages 2018." IEEE Spectrum. https://spectrum.ieee.org/the-2018-top-programming-languages (accessed 5 January 2018).

[20] Open Source Initiative. "Licenses & Standards." https://opensource.org/licenses (accessed 30 July 2021).

[21] Open Source Technology. "Open Source Technology Trends." https://opensource.com/article/17/11/10-open-source-%20technology-trends-2018.%20 (accessed 5 January 2018).

[22] T. B. Do, H. N. H. Nguyen, B. L. L. Mai, and V. Nguyen, "Mining and creating a software repositories dataset," presented at the 7th NAFOSTED Conference on Information and Computer Science (NICS), 2020.

[23] B. Kitchenham and P. Brereton, "A systematic review of systematic review process research in software engineering," *Journal of Information and Software Technology,* vol. 55, no. 12, pp. 2049–2075, 2013.

[24] B. Kitchenham, "Procedures for performing systematic reviews," Department of Computer Science, Keele University, Keele, UK, 2004.

[25] S. Cavanagh, "Content analysis: concepts methods and applications," *Journal of Nurse Researcher,* vol. 4, no. 3, pp. 5–16, 1997.

[26] B. B. Chua, "A survey paper on open source forking motivation reasons and challenges," presented at the Pacific Asia Conference of Information Systems (PACIS), Langkawi, Malaysia, 2017.

[27] B. B. Chua and Y. Zhang, "Applying a systematic literature review and content analysis method to analyse open source developers' forking motivation interpretation, categories and consequences," *Journal of Australasian Information Systems,* vol. 24, no. 1, pp. 1–19, 2020.

[28] J. Biolchini, P. Mian, A. Natali, and G. Travassos, "Systematic review in software engineering," Rio de Janeiro, Brazil, 2005.

[29] C. Okoli and K. Schabram, "A Guide to Conducting a Systematic Literature Review of Information Systems Research," doi: 10.2139/ssrn.1954824.

[30] L. H. Salazar, T. C. Lacerda, J. V. Nunes, and C. G. von Wangenhelm, "Systematic literature review on usability heuristics for mobile phones," *International Journal of Mobile Human Computer Interaction,,* vol. 5, no. 2, pp. 50–61, 2015.

[31] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Keele University and Durham University, 2007.

[32] G. Gousios, B. Vasilescu, S. Bogdan, A. Serebrenik, and A. Zaidman, "Lean GHTorrent: GitHub data on demand," presented at the 11th

Working Conference on Mining Software Repositories (MSR 2014), The Netherlands, 2014.
[33] "Alexa." www.alexa.com/siteinfo. 2017 (accessed 1 October 2017).
[34] K. H. Fung, A. Aurum, and D. Tang, "Social forking in open source software: an empirical study," *CAiSE Forum,* pp. 50–57, 2012.
[35] H. Hsiu-Fang and S. E. Shannon, "Three approaches to qualitative content analysis," *Journal of Qualitative Health Research,* vol. 15, no. 9, pp. 1277–1288, 2005, doi: 10.1177/1049732305276687.
[36] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in GitHub: transparency and collaboration in an open software repository," presented at the ACM 2012 Conference on Computer Supported Cooperative Work, Washington, USA, 2012.
[37] A. Murgia, P. Tourani, B. Adams, and M. Ortu, "Do developers feel emotions? An exploratory analysis of emotions in software artifacts," presented at the 11th Working Conference on Mining Software Repositories (MSR 2014), 2014.
[38] Wikipedia. "MariaDB." https://en.wikipedia.org/wiki/MariaDB (accessed 17 February 2017).
[39] "Merriam-Webster." https://www.merriam-webster.com/ (accessed 1 October 2017).
[40] N. A. Ernst, S. Easterbrook, and J. Mylopoulos, "Code forking in open-source software: A requirements perspective," *arXiv,* vol. 1004.2889, pp. 1–15, 2010.
[41] K. E. Rosengren, "Advances in Scandinavia content analysis: An introduction," in *Advances in Content Analysis*, K. E. Rosengren Ed. Beverly Hills, CA: SAGE, 1981, pp. 9-19.
[42] M. Biazzini and B. Baudry, "May the fork be with you : novel metrics to analyze collaboration on GitHub," presented at the 5th International Workshop on Emerging Trends in Software Metrics, Hyderabad, India, 2014.
[43] H. Fujita and F. Ikuine, "Open source, a phenomenon of generation changes in software development: the case of Denshin 8 Go," *Annals of Business Administrative Science,* vol. 13, no. 1, pp. 1–15, 2014.
[44] L. Nyman, T. Mikkonen, J. Lindman, and M. Fougère, "Perspectives on code forking and sustainability in open source software," presented at the IFIP International Conference on Open Source Systems: Long-Term Sustainability, Buenos Aires, Argentina, 2012.
[45] G. V. Neville-Neil, "Kode vicious: Think before you fork," *Journal of Communications of the ACM,* vol. 54, no. 6, pp. 34–35, 2011, doi: 10.1145/ 1953122.1953137.
[46] L. Nyman, "Hackers on forking," presented at the International Symposium on Open Collaboration, New York, USA, 2014.
[47] B. Ray and M. Kim, "A case study of cross-system porting in forked projects," presented at the 20th ACM SIGSOFT International Symposium on the Foundation of Software Engineering, New York, USA, 2012.
[48] B. B. Chua, "Detecting sustainable programming languages through forking on open source projects for survivability," presented at the IEEE International Symposium on Software Reliability Engineering (ISSRE) in conjunction with a WOSAR workshop, Gaithersburg, USA, 2015.

[49] S. O. Hars, "Working for free? Motivations for participating in open source projects," *International Journal of Electronic Commerce,* vol. 6, no. 3, pp. 25–39, 2002.
[50] K. J. Stewart and S. Gosain, "The impact of ideology on effectiveness in open source software development teams," *Journal of MIS Quarterly,* vol. 30, no. 2, pp. 291–314, 2006.
[51] J. Lerner and J. Tirole, "Some simple economics of open source," *Journal of Industrial Economics,* vol. 50, no. 2, pp. 197–234, 2002.
[52] S. K. Shah, "Motivation, governance and the viability of hybrid forms in open source software development," *Journal of Management Science,* vol. 52, no. 7, pp. 1000–1014, 2006.
[53] E. von Hippel and G. von Krogh, "Open source software and the 'private collective' innovation model: Issues for organization science," *Journal of Economics, Computer Science,* vol. 14, pp. 209–223, 2003.
[54] S. Goode, "Something for nothing: Management rejection of open source software in Australia's top firms," *Journal of Information and Management,* vol. 42, no. 5, pp. 669–681, 2005.
[55] S. Goode, "Exploring organizational information sharing in adopters and non-adopters of open source software: Evidence from six case studies," *Journal of Knowledge and Process Management,* vol. 21, no. 1, pp. 78–89, 2014.
[56] V. N. Subramanian, I. Rehman, M. Nagappan, and R. G. Kula, "Analyzing first contributions on GitHub: What do newcomers do?," *IEEE Software,* 2020, doi: 10.1109/MS.2020.3041241.
[57] The Apache Foundation. "The Apache Foundation." https://www.apache.org (accessed.
[58] M. Gerosa *et al.*, "The shifting sands of motivation: revisiting What drives contributors in open source," presented at the IEEE/ACM 43rd International Conference on Software Engineering (ICSE), 2021.
[59] Wikipedia. "GItHub." https://en.wikipedia.org/wiki/GitHub (accessed 29 January 2019).
[60] D. Celinska, "Coding together in a social network: Collaboration among GitHub users," presented at the 9th International Conference on Social Media and Society (SMSociety '18), 2018.
[61] H. Borges, A. Hora, and M. T. Valente, "Predicting the popularity of GitHub repositories," presented at the 12th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE), 2016.
[62] T. Siddiqui and A. Ahmad, "Data mining tools and techniques for mining software repositories: A systematic review," presented at the Big Data Analytics Conference, 2018.
[63] Y. X. Teng, "A survey of mining software repositories in social network," *Journal of Software,* vol. 15, no. 2, pp. 62–67, 2020.
[64] W. Q. Zhang, L. M. Nie, J. He, Z. Y. Chen, and J. Liu, "Developer social networks in software engineering: construction analysis and applications," *Journal of Information Science,* vol. 57, no. 12, pp. 1–23, 2014.
[65] C. Hora and M. T. Valente, "Understanding the factors that impact the popularity of GitHub repositories," presented at the IEEE International Conference on Software Maintenance and Evolution (ICSME), 2016.

[66] R. Li, P. Pandurangan, H. Frluckaj, and L. Dabbish, "Code of conduct conversations in open source software projects on Github," *ACM Human Computer Interaction,* vol. 5, no. 1, pp. 1–31, 2021, doi: 10.1145/3449093.
[67] C. Vendome, M. Linares-Vasquez, G. Bavota, and M. D. Penta, "License usage and changes: A large-scale study of Java projects on GitHub," presented at the 2015 IEEE 23rd International Conference on Program Comprehension, 2015.
[68] T. S. Heinze, V. Stefanko, and W. Amme, "Mining BPMN processes on GitHub for tool validation and development," in *Enterprise, Business-Process and Information Systems Modeling*. Cham: Springer, 2020, pp. 193–208.
[69] A. Ying, G. Murphy, R. T. Ng, and M. Chu-Caroll, "Predicting source code changes by mining revision history," *IEEE Transactions on Software Engineering,* vol. 30, no. 9, pp. 574–586, 2003.
[70] R. Agrawal, T. Imielinski, and A. N. Swami, "Mining association rules between sets of items in large databases," presented at the International Conference on the Management of Data, 1993.
[71] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," presented at the International Conference on Very Large Data Bases, 1994.
[72] M. Saini and K. K. Chahal, "Change profile analysis of open-source software systems to understand their evolutionary behaviour," *Journal of Frontiers of Computer Science,* vol. 12, no. 6, pp. 1105–1124, 2017.
[73] W. La Cholter, M. Elder, and A. Stalick, "Windows malware binaries in C/C++ GitHub repositories: Prevalence and lessons learned," presented at the 7th International Conference on Information Systems Security and Privacy (ICISSP 2021), 2021.
[74] M. Altherwi and A. Gravell, "Assessing programming language impact on software development productivity based on mining OSS repositories," *Journal of ACM SIGSOFT Software Engineering Notes,* vol. 44, no. 1, pp. 1–3, 2019.
[75] F. Del Bonifro, M. Gabbrielli, A. Lategano, and S. Zacchiroli, "Image-based many-language programming language identification," *Peer Journal of Computer Science,* vol. 7, p. e361, 2021, doi: 10.7717/peerj-cs.631.
[76] S. L. Ramírez-Mora, H. Oktaba, H. G. Adorno, and G. Sierra, "Exploring the communication functions of comments during bug fixing in Open Source Software projects," *Journal of Information and Software Technology,* vol. 136, no. 106584, 2021.
[77] S. Brisson, E. Noei, and K. Lyons, "We are family: Analyzing communication in GitHub software repositories and their forks," presented at the IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2020.
[78] A. Pietri, G. Rousseau, and S. Zacchiroli, "Forking without clicking: On how to identify software repository forks," presented at the 17th International Conference on Mining Software Repositories (MRS '20), 2020.
[79] J. E. Montandon, M. T. Valente, and L. L. Silva, "Mining the technical roles of GitHub users," *Journal of Information and Software Technology,* vol. 131, no. 1064851–12, 2021.

[80] L. Bao, X. Xia, D. Lo, and G. Murphy, "A large scale study of long-time contributor prediction for GitHub projects," *Journal of IEEE Transactions on Software Engineering,* vol. 47, no. 6, pp. 1277–1298, 2021.

[81] V. K. Eluri, T. A. Mazzuchi, and S. Sarkani, "Predicting long-time contributors for GitHub projects using machine learning," *Journal of Information and Software Technology,* vol. 138, no. 106616, 2021.

[82] K. Du *et al.*, "Understanding promotion-as-a-service on GitHub," presented at the Annual Computer Security Applications Conference (ACSAC '20), New York, USA, 2020.

[83] F. Fronchetti, I. Wiese, G. Pinto, and I. Steinmacher, "What attracts newcomers to onboard on OSS projects? TL;DR: Popularity," presented at the 15th IFIP International Conference on Open Source Systems (OSS), Montreal QC, Canada, 2019.

[84] R. Kallis, A. D. Sorbo, G. Canfora, and S. Panichella, "Predicting issue types on GitHub," *Journal of Science of Computer Programming,* vol. 205, no. 3, 2020, doi: 10.1016/j.scico.2020.102598.

[85] R. Kapur and B. Sodhi, "Estimating defectiveness of source code: A predictive model using Github content," *arXiv,* vol. 1803.07764, 2018.

[86] M. O. F. Rokon, R. Islam, A. Darki, E. E. Papalexakis, and M. Faloutsos, "Sourcefinder: Finding malware source-code from publicly available repositories," presented at the 23rd International Symposium on Research in Attacks Intrusions and Defenses, 2020.

[87] H. L. Zhou *et al.*, "GitEvolve: Predicting the evolution of GitHub repositories," *arXiv,* vol. abs/2010.04366, 2020.

[88] S. Weber and J. Luo, "What makes an open source code popular on GitHub?," presented at the IEEE International Conference on Data Mining Workshop, 2014.

[89] B. W. Silverman and M. C. Jones, "E. Fix and J.L. Hodges (1951): An important contribution to nonparametric discriminant analysis and density estimation: Commentary on Fix and Hodges (1951)," *International Statistical,* vol. 57, no. 3, pp. 233–238, 1989, doi: 10.2307/1403796.

[90] T. Cover and P. Hart, "Nearest neighbour pattern classification," *IEEE Transactions on Information Theory,* vol. 13, no. 1, pp. 21–27, 1967, doi: 10.1109/TIT.1967.1053964.

[91] P. Cunningham and S. J. Delany, "K-Nearest Neighbour Classifiers," *Multiple Classifier Systems,* vol. 34, no. 8, pp. 1–17, 2007.

[92] J. Gou, L. Du, Y. Zhang, and T. Xiong, "A new distance weighted k-nearest neighbor classifier," *Journal of Information and Computational Science,* vol. 9, no. 6, pp. 1429–1436, 2011.

[93] J. W. Han, M. Kamber, and J. Pei, "Data mining: Concepts and techniques," in "Data Management Systems," 2000.

[94] D. J. Hand, H. Mannila, and P. Smyth, *Principles of Data Mining*. Cambridge: MIT Press, 2001.

[95] S. Y. Jiang, G. S. Pang, M. L. Wu, and L. M. Kuang, "An improved K-nearest-neighbor algorithm for text categorization," *Journal of Expert Systems with Applications,* vol. 39, no. 1, pp. 1503–1509, 2012.

[96] M. McCord and M. Chuah, "Spam detection on Twitter using traditional classifiers," presented at the International Conference on Autonomic and Trusted Computing, 2011.

[97] K. Odajima and A. P. Pawlovsky, "A detailed description of the use of the kNN method for breast cancer diagnosis," presented at the 7th International Conference on Biomedical Engineering and Informatics (BMEI), 2014.

[98] H. Wang, "Nearest neighbours without K: A classification formalism based on probability," Faculty of Informatics, University of Ulster, Northern Ireland, UK, 2002.

[99] X. Wu *et al.*, "Top 10 algorithms in data mining," *Knowledge and Information Systems,* vol. 14, no. 1, pp. 1–37, 2008, doi: 10.1007/s10115-007-0114-2.

[100] B. B. Chua and Y. Zhang, "Predicting open-source programming language repository file survivability from forking data," presented at the 15th International Symposium on Open Collaboration (OpenSym'19), Skövde, Sweden, 2019.

[101] K. R. Linberg, "Software developer perceptions about software project failure: A case study," *Journal of Systems and Software,* vol. 49, no. 2, pp. 177–192, 1999.

[102] "Octoverse." https://octoverse.github.com/ (accessed 16 February 2019).

[103] M. Montague and J. Aslam, "Relevance score normalization for metasearch," presented at the 10th international conference on Information and knowledge management (CIKM), New York, USA, 2001.

[104] "Fossbytes." https://fossbytes.com/what-programming-language-does-your-country-like (accessed 16 August 2021).

[105] "Security Today." https://securitytoday.com/articles/2019/08/19/the-dangers-of-opensource-vulnerabilities-and-what-you-can-do-about-it.aspx (accessed 16 August 2021).