Applying a Systematic Literature Review and Content Analysis Method to Analyse Open Source Developers' Forking Motivation Interpretation, Categories and Consequences

Bee Bee Chua
Faculty of Engineering and Information Technology
University of Technology, Sydney
choiceno1@mailcity.com

Ying Zhang
Faculty of Engineering and Information Technology
University of Technology, Sydney

Abstract

In open source (OS) environments, forking is a powerful social collaborative technique that creates a social coding community and increases code visibility but it has not been adopted by OS software (OSS) developers. This paper investigates OS forking divergence using contextual frameworks (systematic literature review and content analysis) to analyse OSS developer forking motivation, interpretation, categorisation and consequences. We identified five theoretical forking patterns: 1) forking can revive original project health; 2) few effective frameworks exist to describe project-to-project developer migration; 3) there is a literature on social forking community behaviour; 4) poor guidance is a threat to forking; and 5) most research uses mixed methods. We introduce guidelines for OSS communities to reduce organisational barriers to developer motivation and highlight the important of understanding developer forking. The challenge remains to analyse forking and sustainability from a social community perspective, particularly how programming language, file repositories and developer interest can predict forking motivation and behaviour for both novice OSS developers or experienced developers who want to improve forking performance.

Keywords: Open Source, Forking, Motivation, Sustainability, Systematic Literature Review, Fork Visibility

1 Introduction

GitHub is a hosting website for developing open source software (OSS) through social coding by multiple developers. GitHub stores projects, files, programing languages, licenses and developer profiles. In May 2019, GitHub reported having over 37 million users since its inception, and more than 100 million repositories (including at least 28 million public repositories), making it the largest global host of open source (OS) code (Gousios et al., 2014). GitHub currently has 26 million registered developers from 110,000 organisations and an additional 20 million developers and users visit GitHub daily without registering (Alexa, 2017). 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 (Meyerovich & Rabkin, 2013). 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 development 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 (Glass, 2003; Fung, Aurum & Tang, 2012; Gamalielesson & Lundell, 2013; Fujita & Ikuine, 2014; Jiang, Lo, He, Xia, Singh & Zhang, 2016; Azarbakht & Jensen, 2017), 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.

To the best of our knowledge, 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 objective of this research was therefore to identify types of developer forking motivation and forking consequences cited in the existing OS literature through a systematic literature review (SLR) adopted from Biolchini, J et al. (2005) 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) from Hsieh and Shannon (2016) 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 paper include: 1) summarising the existing evidence base on forking motivation and consequences into a methodological framework; 2) providing a reference check 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.

The paper is organised as follows: section 2 discusses the research study motivation and research questions; section 3 describes the SLR, content analysis methods and the proposed framework; section 4 presents the findings (forking interpretations and response to the research questions); then section outlines conclusions and possible future research directions.

2 Research Study Motivation and Research Questions

2.1 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 (Dabbish, Stuart, Tsay & Herbsleb, 2012; Fung et al., 2012; Robles & Gonzalez-Barahona, 2012; Jiang et al., 2016) 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 license, 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. (2014) noted that developers feel emotions about OSS artefacts, such as joy, love, anger, surprise, sadness and fear.

2.2 Research questions

Forking is the creation of a new software repository by copying another repository (Jiang et al., 2016). 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 (Wikipedia, 2001).

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 (Meyerovich & Rabkin, 2013). The number of terminated projects is also increasing due to low sustainable community participation and collaboration to fix bugs and improve features (Jiang et al., 2016). 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 (Merriam-Webster Dictionary, 1999). 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) started in 1990 with support from many of the world's largest OSS projects and contributors, including Debian, Drupal Association, FreeBSD Foundation, Linux Foundation, Mozilla Foundation, Wikimedia Foundation and WordPress Foundation (Open Source Initiative, 1990). The aim was to uphold the OSI's mission and Open Source Definition through the OSI Affiliate Agreement (OSI Affiliate Agreement). 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 (Jiang et al., 2016; Fung et al., 2015). 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 (Ernst, Easterbrook & Mylopoulos, 2010). 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.

3 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 (Biolchini, Mian, Natali & Travassos, 2005; Okoli & Schabram, 2010; Salazar, Lacerda, Nunes & von Gresse, 2013). Software engineering researchers (Kitchenham, 2004; Kitchenham & Charters, 2007; Kitchenham & Brereton, 2013) 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.

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 (Figure 1), as it is a flexible method for analysing text data, with approaches ranging from impressionistic, intuitive and interpretive to systematic and strict textual analyses (Cavanagh, 1997; Rosengren, 1981). Highly cited content analysis researchers Hsieh and Shannon (2016) defined three approaches: 1) a conventional analysis where coding categories are derived directly from the text data; 2) a directed approach where user analysis begins with a theory or relevant research findings as guidance for initial codes; and 3) summative content analysis that involves counting and comparing keywords or content followed by interpreting the underlying context.

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 (Figure 1).

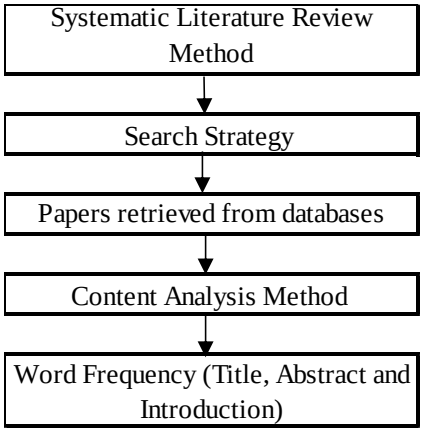

Figure 1. Combined approaches: systematic literature review and content analysis methods

3.1 Systematic literature review search criteria

To ensure the literature search was specific and to identify the most relevant, high-quality articles, the inclusion criteria were:

1. Peer-reviewed conference or journal papers, published and indexed either in Google Scholar, ACM, IEEE, Science Direct, Springer or MISQ; AND
2. Written in English; AND
3. 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

4. Published from 1990 to 2017; AND
5. Published from top quality Information Systems Conferences or Journals; AND
6. 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.

3.2 Search strategy

Two approaches were applied to conduct the SLR search (Figure 2). The first search was conducted on 1 October 2017 on Google Scholar for the term "open source forking behaviour", resulting in 21,200 URLs. Results were then sorted by relevance and filtered for papers published from 1996 to 2017, resulting in 9,530 URLs. These papers were both peer-reviewed and non-peer-reviewed, spanning a variety of disciplines, from economics, management and software engineering through sociology (Biolchini et al., 2005; Okoli & Schabram, 2010; Salazar et al.; 2013). 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 (Table 1).

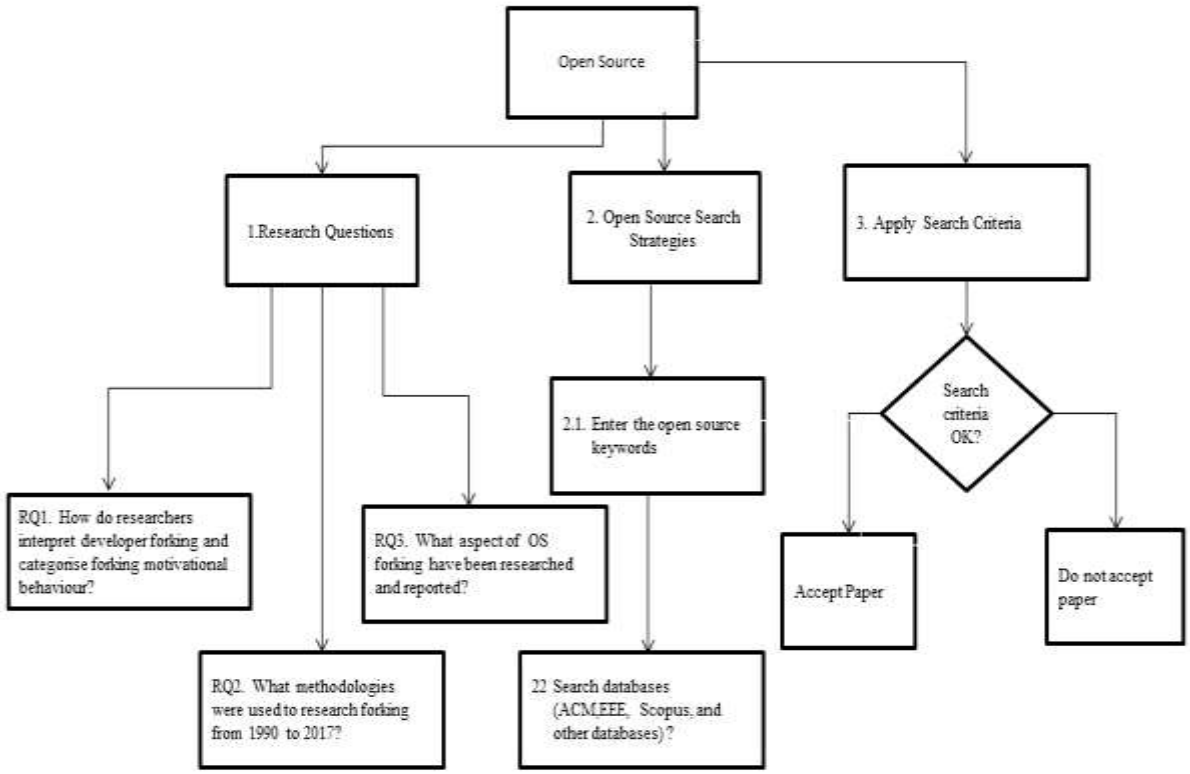

Figure 2. The systematic literature review search strategy for research papers

Database	Number	Authors
Google Scholar	8	Biazzini & Baudry (2014); Moen (1999); Ernst et al. (2010); Ikuine & Fujita (2014); Fujita & Ikuine (2014); Fung et al. (2012); Gamalielesson & Lundell (2013); Nyman, Mikkonen, Lindman & Fougère (2012)
ACM	6	Glass (2003); Neville-Neil (2011); Dabbish, et al. (2012); Ray & Kim (2012); Nyman (2014); Ray, Posnett, Filkov & Devanbu (2014)
IEEE	2	Chua (2015); Cosentino, Javier, Izquierdo & Cabot (2017)
MISQ	1	Krogh, Haefliger, Spaeth & Wallin (2012)
Springer	4	Robles & Gonzalez-Barahona (2012); Azarbakht & Jensen (2017); Jiang et al. (2016); Nyman & Mikkonen (2011)

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

3.3 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 (Table 2). 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.

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

Table 2. A forking motivation methodological framework

3.4 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 (Table 3). For example, when reviewing the papers "Code Forking in Open-Source Software: A Requirements Perspective" (Ernst et al., 2010) and "Perspective on Code Forking and Sustainability in Open Source Software" (Nyman et al., 2012) the word 'code' occurred twice so '2' was entered under 'code' forking type identified by the Google Scholar search in Table 2. 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.

Forking type	Paper identified via					TOTAL
	ACM	IEEE	Springer	MISQ	Google Scholar	
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

Table 3 Forking interpretation types

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

1. *Developer forking interpretations*: 7 interpretations of forking (Table 2).
2. *Developer motivation and reasons*: a subset of papers reported similar variables (Table 3). For instance, Krogh et al. (2012), Fung et al. (2012), Glass (2013) and Jiang et al. (2016) reported divergent specialisation, objective misalignment, poor governance and leadership and culture.

3. *Forking sustainability*: four groups of researchers (Ernst et al., 2010; Nyman et al., 2012; Gamalielesson et al 2013; Jiang et al., 2016) undertook real-world projects, comparing original versus forked projects (Table 3). 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.

4. *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 (Moen, 1999; Glass, 2003; Neville-Neil, 2011; Ikuine & Fujita, 2014; Fujita & Ikuine, 2014; Cosentino et al., 2017; Azarbakht & Jensen, 2017). Neville (2011) pointed out that technical developers' roles are becoming specialised.

4 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, license 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.

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

These findings reveal a diversity of forking interpretations (Table 3), 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 Table 4).

4.1.1 Open source forking

The early 1990s saw a proliferation of research on OS motivation. Krogh and colleagues (2012) reviewed seven years of publications and identified 40 papers that focused on OS developer motivation, including Hars (2002), Stewart & Gosain (2006), Hertel et al. (2003), Lerner & Tirole (2002) and Shah (2006). 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. Extrinsic motivation may include being paid for the work or finding a career in coding. Hippel & Krogh (2003) and Goode (2005; 2014) studied organisational information sharing in adopters and non-adopters of OSS and innovation models as influencing factors on motivation. They found more reputable organisations and innovative projects are more likely to attract OSS developer attention to download or copy repository files.

4.1.2 Project forking

Nyman and Mikkonen (2011) defined that a project fork takes place when software developers copy source code from one software package and use it to begin an independent development work. In general, forking results in an independent version of the system that is maintained separately from its origin. Nyman and Mikkonen (2011) 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 provided real-life examples of current high profile OS projects that either started from a fork or were common targets for forking.

4.1.3 Software forking

Ikuine and Fujita (2014) referred to software forking as the continuous development of software, by the original developer or others. When other developers take over, the original developer must share the source code. Software forking focuses on the product itself, such as Microsoft software, Facebook software and email applications.

4.1.4 Social forking

Fung, Aurum and Tang (2012) 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 7,000 developers made approximately 8,000 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 artefacts or from multiple predecessors' artefacts. A branch in turn is a copy of all the predecessor's artefacts (Fung et al., 2012).

4.1.5 Code forking

Code forking is defined as a forked project copied from existing code base and moved in a direction different from the project leadership. Forking the code base allows developers to leverage existing functionality while also addressing new requirements. Although flexible, forking has inherent difficulties, such as maintenance, evolution, and social factors concerning the development community. A broad definition of a code fork is when the code from an existing program serves as a fork (Nyman et al., 2014); it is the basis for a new version of the program, more specifically, a version that seeks to continue to exist apart from the original.

4.1.6 Programming language forking

Chua (2015) 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 (2019) then proposed three forking pattern types ('once-only', intermittent or steady) and potential reasons behind short-lived programming languages.

4.1.7 File repository forking

A file repository fork is mainly used to make contributions to original repositories and is beneficial for the OSS community (Jiang et al., 2016). Actions such as submitting pull requests, fixing bugs, adding new features and keeping copies are motivations for developers to fork repositories. A repository written in a developer's preferred programming language is more likely to be forked and developers mostly fork repositories from creators. Attractive repository owners include organisations, as they have more followers.

Type	Interpretation	Studies	Citing authors within paper set
Forking motivation			
Coding for revising requirements	Requirement change	Ernst et al. (2010)	Ernst et al. (2010) cited by Fung et al. (2012); Jiang et al. (2016)
Seeking a coding job	Recruitment of contributors	Biazzini & Baudry (2014)	Nil
Licensing compliance	Licensing compliance	Biazzini & Baudry (2014); Dabbish et al. (2012); Jiang, et al. (2016)	Nil
Software compliance	Software interoperability	Krogh et al. (2012); Meyerovich, & Rabkin (2013); Nyman (2014); Tegawendé, Bissyandé, Thung, Lo, Jiang & Réveillère (2013)	Nil
Reviving original project development duration	Cessation of original project	Nyman (2014); Robles & Gonzalez-Barahona (2012); Ray & Kim (2012); Tegawendé et al. (2013); Chua (2015)	Nyman (2014) cited by Jiang et al. (2016)
Extending community social coding development	More community driven development	Dabbish et al. (2012)	Ray et al. (2014) cited by Jiang et al. (2016)
Ownership implication	Legal implication on ownership and conflict over brand ownership	Fung, Aurum & Tang et al. (2012); Nyman (2014); Nyman & Mikkonen (2011); Ray & Kim (2012)	
Business strategy risk	Commercial strategy forks	Dabbish et al. (2012)	
Team coding skill inequality	Differences among developer team	Nyman (2014)	
Community socialisation	Building new community through social interaction, sharing and collaboration	Dabbish et al. (2012); Fung et al. (2012); Robles, & Gonzalez-Barahona (2012)	
Coding by socialising	Social network coding	Jiang et al. (2016); Fung et al. (2012)	
Divergent specialisation	New specialisation, divergent technical views	Nyman (2014); Nyman & Mikkonen (2011); Ray & Kim (2012)	Nil
Objective misalignment	Different technical objectives		
Poor leadership	Poor project governance	Nyman (2014); Nyman & Mikkonen (2011); Robles & Gonzalez-Barahona (2012)	
Culture trait	Cultural differences		

Type	Interpretation	Studies	Citing authors within paper set
Software activity	Project specialty to generate commits	Ray & Kim (2012); Tegawendé et al. (2013)	
Ecosystem	System between system sharing resources and infrastructure		
Forking sustainability			
Community activity	Communities retention	Ernst, et al. (2010); Gamalielesson & Lundell (2013). Jiang et al. (2016); Nyman et al. (2012); Azarbakht & Jensen (2017); Cosentino et al. (2017)	Ray et al. (2014) cited by Jiang et al. (2016); Gamalielesson & Lundell (2013)
Forking lessons learnt			
No formal process	No guidance/ direction	Ikuine & Fujita (2014); Fujita & Ikuine (2014); Azarbakht & Jensen (2017)	Nil
Legal implication	Copyright	Glass (2003); Azarbakht & Jensen (2017)	
	Licensing conflict	Moen (1999); Azarbakht & Jensen (2017)	
Transfership	Project ownership	Ikuine & Fujita (2014); Fujita & Ikuine (2014); Cosentino et al. (2017); Azarbakht & Jensen (2017)	
Product expertise shortage	Technical developers become product expert	Neville-Neil (2011)	
Upgrade of developer role to product role	Role movement	Glass (2003); Ikuine & Fujita (2014); Cosentino et al. (2017)	Glass (2003) cited by Fung et al. (2012)
Community divide	Divide community fork	Azarbakht & Jensen (2017); Cosentino et al. (2017)	Nil

Table 4. Fork categorisation, sustainability and lessons learnt

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

Figure 4 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).

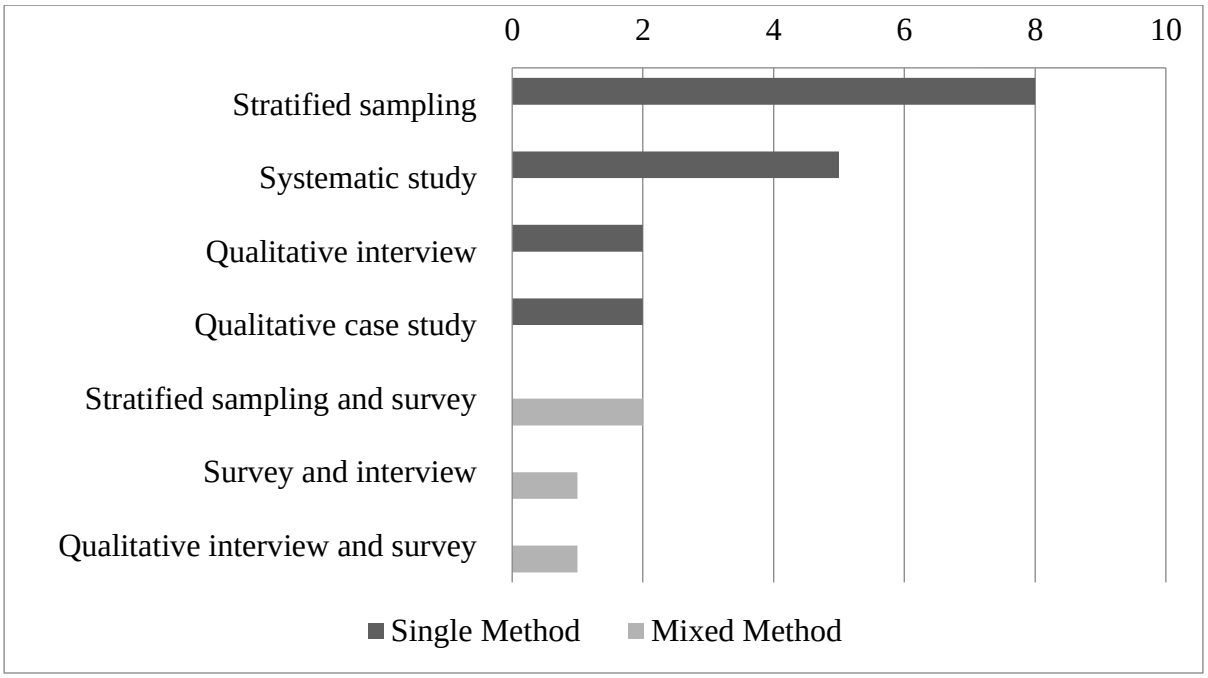

Figure 3. Data collection methods in the 21 papers

4.3 What aspects of OS forking have been researched and reported?

Figure 4 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).

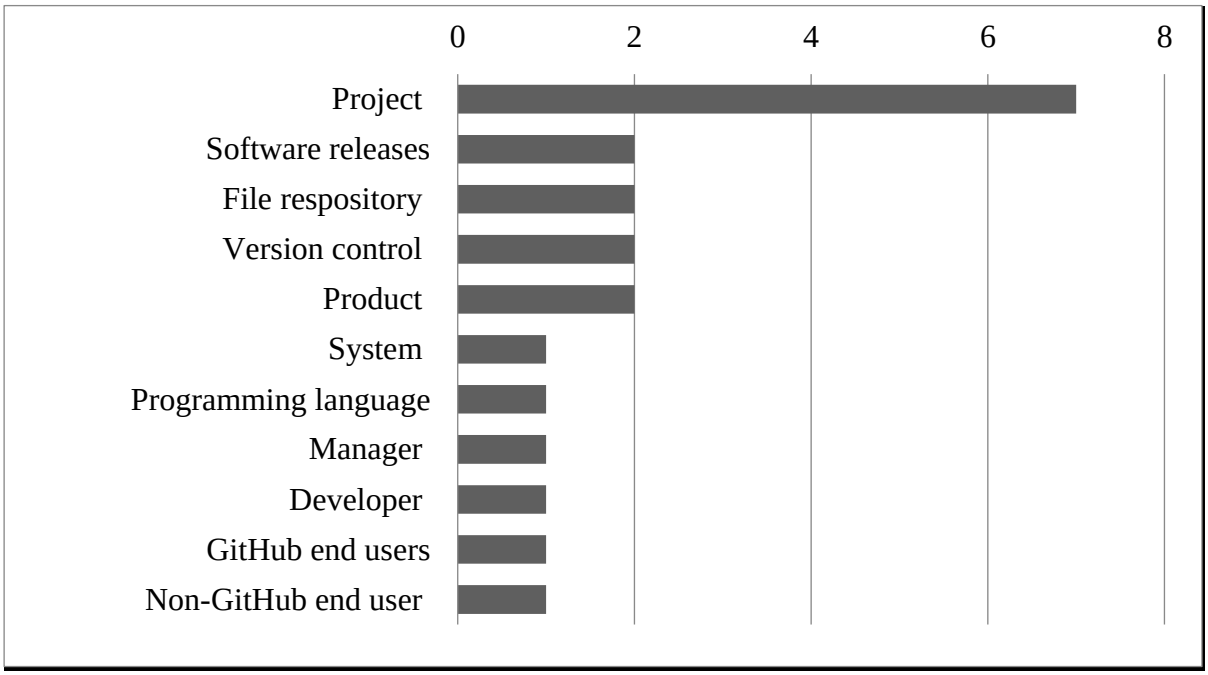

Figure 4. Units of analysis in the 21 papers

Figure 5 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.

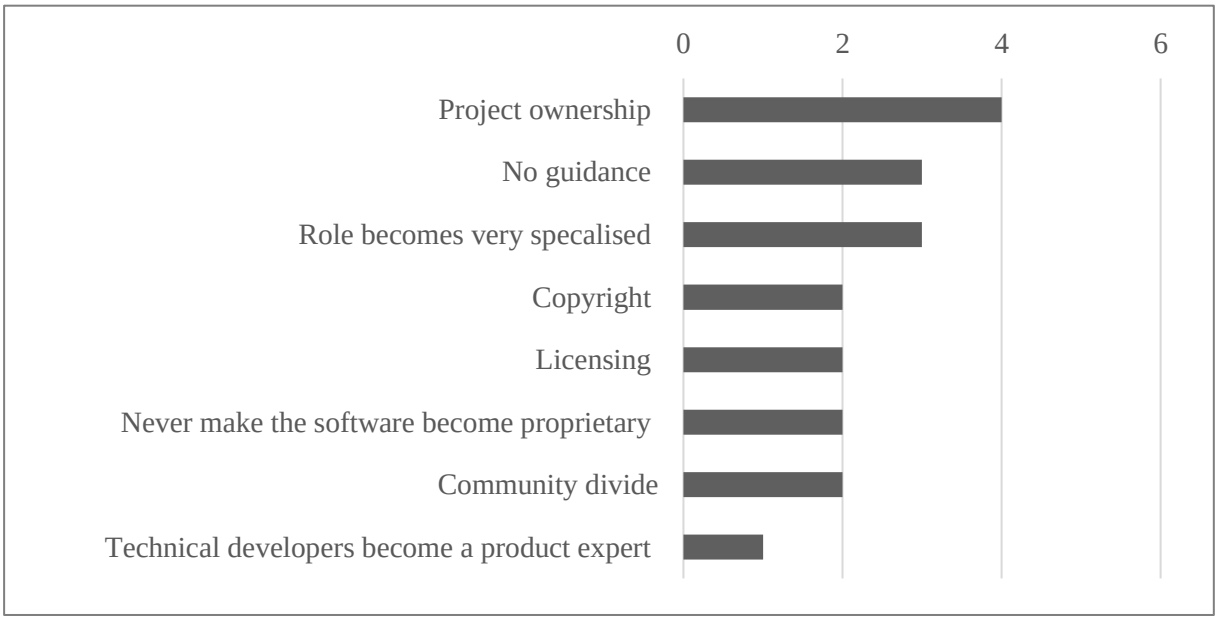

Figure 5. Forking lessons learnt across the 21 papers

5 Conclusions and Future Work

Forking is one of the most critical technique in OS research today. Our analysis of 21 papers 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 the project or terminate the file repository.

To the best of our knowledge, 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.

This paper provides a quick reference for OSS researchers to understand categories of developer forking motivation, introduce guidelines for OSS communities on ways to reduce organisational barriers to developer motivation, and, most importantly, highlight that new or existing project sponsors should focus on understanding developer forking motivation, to positively influence achieving a healthy source code.

This study also identifies some challenging areas for future work.

1. **Append new findings into the body of knowledge on OS forking behaviour**. Applying the combined approaches of SLR and CAM 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 programing 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 & Jensen (2017) adopted a developer-oriented statistical approach to determine what causes people in complex software development networks to decide to fork (break away), and what changes a community goes through when deciding to divide Different or conflicting goals, communication 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**. Valentio, Javier, Izquierdo and Cabot (2017) used a SLR to show that forking is a good indicator of project longevity and the chance of forking is highly dependent on the project, where developers provide additional contact information (e.g., emails, personal website URLs that are clearly active or aligned with popular project owners) to increase social connections between a project owner and forker, and increase developer community size for medium-size projects and projects that are written in a forker's preferred programming language. 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.

References

Alexa, (2017). https://www.alexa.com/siteinfo. (Accessed on 1 October 2017)

Azarbakht, A.E., & Jensen, C. (2017). Longitudinal analysis of the run-up to a decision to break-up (fork) in a community. *Proceedings of the IFIP International Conference on Open Source Systems.* OSS 2017, Springer.

Biazzini, M., & Baudry, B. (2014). "May the fork be with you": novel metrics to analyze collaboration on GitHub. *Proceedings of the 5th International Workshop on Emerging Trends in Software Metrics*, June 2014, Hyderabad, India.

Biolchini, J., Mian, P. G., Natali, A. C., & Travassos, G. H. (2005). *Systematic Review in Software Engineering*. Technical Report RT-ES 679/05, COPPE/UFRJ, Rio de Janeiro, Brazil.

Cavanagh, S. (1997). Content analysis: concepts, methods and applications. *Nurse Researcher 4*(3), 5–16.

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

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

Chua, B and Zhang Y. (2019). Predicting open source repository by programming language survivability from forking data. *The 15th International Symposium on Open Collaboration (Opensym)*, August 20–22, 2019, Skövde, Sweden. Association for Computing Machinery (ACM).

Dabbish, L., Stuart, C., Tsay, J., & Herbsleb, J. (2012). Social coding in GitHub: transparency and collaboration in an open software repository. *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*

Ernst, N. A., Easterbrook, M. A., & Mylopoulos, J. (2010). Code forking in open-source software: a requirements perspective. *CoRR*, abs/1004.2889.

Fujita, H., & Ikuine, F. (2014). Open source, a phenomenon of generation changes in software development: the case of Denshin 8 Go. *Annals of Business Administrative Science, 13*(1), 1–15.

Fung, K. H., Aurum, A., & Tang, D. (2012). Social forking in open source software: an empirical study. *CAiSE Forum*, 50–57.

Gamalielesson, J., & Lundell, B. (2013). Sustainability of open source software communities beyond a fork: how and why has the LibreOffice project evolved? *Journal of Systems & Software, 89*, 128–145.

Glass, R. L. (2003). A sociopolitical look at open source. *Communications of the ACM 46*(11), 21–23.

Goode, S. (2005). Something for nothing: management rejection of open source software in Australia's top firms. *Information & Management, 42*(5), 669–681.

Goode, S. (2014). Exploring organizational information sharing in adopters and non-adopters of open source software: Evidence from six case studies. *Knowledge and Process Management, 21*(1), 78–89.

Gousios, Georgios; Vasilescu, Bogdan; Serebrenik, Alexander; Zaidman, Andy. (2014) *Lean GHTorrent: GitHub Data on Demand.* The Netherlands: Delft University of Technology & Eindhoven University of Technology.

Hars, S. O. (2002). Working for free? Motivations for participating in open-source projects. *International Journal of Electronic Commerce, 6*(3), 25–39.

Hippel, E. V., & Krogh, G. V. (2003). Open source software and the "private-collective" innovation model: Issues for organization science. *Organizational Science, 14*(2), 209–223

Hertel, G., Niedner, S., & Herrmann, S. (2003). Motivation of software developers in open source projects: An internet-based survey of contributors to the Linux Kernel. *Research Policy, 32*(7), 1159–1177.

Hsiu-Fang H., & Shannon, S. E. (2016). Three approaches to qualitative content analysis. *Journal of Qualitative Health Research, 15*(9), 1277–1288.

Ikuine, F., & Fujita, H. (2014). How to avoid fork: the guardians of Denshin 8 Go, Japan. *Annals of Business Administrative Science, 13,* 283–298.

Jiang, J., Lo, D., He, J. J., Xia, X, Singh P. K., & Zhang, L. (2016). Why and how developers fork what from whom in GitHub. *Journal of Empirical Software Engineering, 100*(21), 1–32.

Kitchenham, B. (2004). *Procedures for Performing Systematic Reviews*, Technical Report. TR/SE-0401, Department of Computer Science, Keele University, Keele, UK.

Kitchenham, B., & Brereton, P. (2013). A systematic review of systematic review process research in software engineering. *Information & Software Technology, 55*(12), 2049–2075.

Kitchenham, B., & Charters, S. (2007). *Guidelines for Performing Systematic Literature Reviews in Software Engineering,* Joint Report EBSE 2007-001, Keele University and Durham University.

Krogh, V. G., Haefliger, S., Spaeth, S., & Wallin, M. W. (2012). Carrots and rainbows: motivation and social practice in open source software development. *MIS Quarterly, 36*(2), 649–676.

Lerner, J., & Tirole, J. (2002). Some simple economics of open source. *Journal of Industrial Economics, 50*(2), 197–234.

MariaDB. In *Wikipedia.* https://en.wikipedia.org/wiki/MariaDB. Retrieved 17 February 2017.

Meyerovich, L. A., & Rabkin, A.S. (2013). Empirical analysis of programming language adoption. *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*, October 29–31, 2013. Indianapolis, Indiana, USA.

Moen, R. (1999). *Fear of Forking.* http://linuxmafia.com/faq/Licensing_and_Law/forking.html. Retrieved 22 November 2016.

Murgia, A., Tourani P., Adams, B., & Ortu, M. (2014). Do developers feel emotions? An exploratory analysis of emotions in software artifacts. *Proceedings of the 11th Working Conference on Mining Software Repositories*, 262271.

Neville-Neil, G. V. (2011). Kode vicious: think before you fork. *Communication of the ACM (54)*, 34–35. doi: 10.1145/ 1953122.1953137

Nyman, L. (2014). Hackers on forking. *Proceedings of the International Symposium on Open Collaboration*, ACM, New York, NY, USA, ISBN: 978-1-4503 30169. 4–12.

Nyman, L., & Mikkonen, T. (2011). To fork or not to fork: fork motivations in SourceForge projects. *International Journal of Open Source Software & Processes, 3*(3), 1–9.

Nyman, L., Mikkonen, T., Lindman, J., & Fougère, M. (2012). Perspective on code forking and sustainability in open source software. *Proceedings of the IFIP International Conference on Open Source Systems,* Open Source Systems: Long-Term Sustainability, 274–279.

Okoli, C., & Schabram, K. (2010). A guide to conducting a systematic literature review of information systems research. Available at SSRN: dx.doi.org/10.2139/ssrn.1954824

Open Source Initiative. (1990). https://opensource.org/node/755. Retrieved 25 November 2016.

Open Source Initiative Affiliate Agreement. (1998) www.opensource.org. Retrieved 25 November 2016.

Ray, B., & Kim, M. (2012). A case study of cross-system porting in forked project. *Proceedings of the 20th ACM SIGSOFT International Symposium on the Foundation of Software Engineering.* November 2012.

Ray, B., Posnett, D., Filkov, V., & Devanbu, P. (2014). A large scale study of programming languages and code quality in GitHub. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, November 16–21 2014. Hong Kong, China.

Ray, B., Wiley, C., & Kim, M. (2012). Repertoire: a cross-system porting analysis tool for forked software project. *Proceedings of the 20th ACM SIGSOFT International Symposium on the Foundation of Software Engineering*, November 2012.

Robles, G., & Gonzalez-Barahona, M. (2012). A comprehensive study of software forks: dates, reasons and outcomes. Open source systems: long-term sustainability. *IFIP Advances in Information & Communication Technology*, *378*(1), 1–14.

Rosengren, K. E. (1981). Advances in Scandinavia content analysis: an introduction. In *Advances in Content Analysis,* K. E. Rosengren (ed.), Beverly Hills, CA: Sage, 9–19.

Salazar, L. H. A., Lacerda, T., Nunes, J. V., & von Gresse, C. (2013). Systematic literature review on usability heuristics for mobile phones. *International Journal of Mobile Human Computer Interaction, 5*(2), 12–22.

Shah, S. K. (2006). Motivation, governance, and the viability of hybrid forms in open source software development. *Management Science, 52*(7), 1000–1014.

Stewart, K. J., & Gosain, S. (2006). The impact of ideology on effectiveness in open source software development teams. *MIS Quarterly, 30(2),* 291-314

Tegawendé, F., Bissyandé, T., Thung, F., Lo, D., Jiang, L.X., & Réveillère, L. (2013). Popularity, interoperability, and impact of programming languages in 100,000 open source projects. *Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual Conference 23-26 July, Kyoto, Japan*

Valentio. C, Javier, L. Izquierdo, C. Cabot, J. (2017). A systematic mapping study of software development with GitHub. *Access IEEE*, *5*(1), 7173–7192.

doi: https://doi.org/10.3127/ajis.v24i0.1714