The Impact of Free and Open Source Software(FOSS) on Developers' Productivity

Maimuna Badjie

Field of study: Information Systems

Credits: 30 credits

Presented: Spring 2022

Supervisor: Steve McKeever

Department of Informatics and Media

Uppsala University

Abstract

Despite the abundance of literature on free and open source software (FOSS), its impact on individual developers' productivity is a topic that is under explored. Moreover, the traditional measures of developers' productivity are based on objective measures (lines of code) instead of subjective measures (self-rated productivity). To bridge this gap and add new knowledge to existing literature, the study explored the impact of FOSS on developers' productivity based on self-rated productivity measure. Hence, the study used an exploratory approach of qualitative research, a theoretical framework and interviews as a method of data collection. The findings indicate that FOSS actually has a positive impact on developers' productivity and the rate of adoption is high among young and experienced developers. However, the risks and security challenges that come with FOSS based on its open model actually creates room for lack of trust and thus can reduce the adoption rate. Though FOSS has improved developers' productivity when it comes to software development, the issues of poor documentation and lack of documentation encountered when trying to modify or enhance FOSS actually hinder developers' productivity. Hence the free and open source software community should come up with ways of tackling these challenges in order for the developers to be more productive.

Keywords

Free Software, Open Source Software (OSS), Free and Open Source Software (FOSS), Feller and Fitzgerald framework, Productivity, Self-Rated Productivity (Perceived Productivity)

Dedication

This thesis is Dedicated to my Brother Modou Badjie for always being my support system through out this whole journey.

Acknowledgement

First of all, I would like to thank God for showering me with His enormous blessings and guiding me to find my way through all the difficulties I have experienced.

I would also like to thank The Uppsala University President's Club for awarding me with the scholarship because my master's studies in Sweden would not have been possible without this opportunity.

Moreover, I would like to thank my supervisor Steve Mckeever who made this work possible. His support and guidance carried me through all the stages of writing this thesis. I would also like to extend my sincere gratitude to the interviewees for the time allocated to share their knowledge and experience on the topic which helps facilitate the analysis process.

Special thanks to my Mother, Father, Brother and entire family for the continuous prayers and support throughout this journey. A big thank you to all my friends and loved ones for the continuous motivation, encouragement and being my support system.

Table of Contents

List of Tables

1. Introduction

In the famous book The Cathedral and the Bazaar, the author Eric Raymond identified what is happening in the Linux world as a peer-to-peer decentralized market bazaar-like style. In this style, there are very short release intervals, constant feedback from the community and intense peer-review process. According to Raymond (2001), this bazaar-like style is embraced by the free and open source software development community.

Free and open source software (FOSS) is software that is both free and open source where anyone is freely licensed to use, copy, study, modify or enhance the software in any way. The source code is openly accessible so that people are encouraged to voluntarily improve the design and make the software evolution easy. Free and open source software allow developers to enhance the program's performance, add some features and fix errors.

On the other hand, productivity is commonly defined as a ratio of output to input use. While there is no disagreement on this general notion, the productivity literature and its various applications revealed that there is neither a unique definition nor a single measure of productivity (Giovannini, & Nezu, 2001; Umar et al., 2021). Moreover, some of the measures of productivity have changed over the years. Measuring a developers' productivity is quite challenging as it is likely that no single measure will do. The traditional objective measures of developers' productivity, like number of commits and lines of code tend to be ineffective due to the qualitative nature of the output produced (Prasad et al., 2002; Palvalin, 2019). Hence due to the obstacles in achieving objective measures, subjective measures like self-rated productivity or peer evaluation are widely used.

1.1 Knowledge Gap

A lot of research on free and open source software phenomena has been conducted in previous years. However, the impact of free and open source software on developers' productivity is a topic that is under explored. Moreover, knowledge on the adoption of FOSS by individual developers and how it impacts their productivity is very limited as literatures in this area focus on the organization level (Nagy et al., 2010). As free and open source software is increasingly used as a key input by individuals and organizations, understanding its impact on individual developers' productivity becomes quite relevant. Moreover, some of the measures of productivity have changed over the years. According to Prasad et al., (2002) and Palvalin (2019), the traditional objective measures of developers' productivity,

like number of commits and lines of code tend to be ineffective due to the qualitative nature of the output produced. Hence, exploring the impact of FOSS on developers' productivity based on subjective measures is of great relevance.

1.2 Purpose

This study is set to explore the impact of free and open source software on developers' productivity. Hence, the study tends to use self-rated productivity measure through interviews to understand what impact FOSS has on developers' productivity. Understanding the impact of FOSS on developers' productivity does not only help to bridge the knowledge gap and contribute to existing literature on productivity, but also shows that FOSS is no longer a rare phenomenon since it is used as a key input by developers and thus can have a negative or positive impact on their productivity.

Therefore, the study seeks to contribute new knowledge to the body of knowledge in information systems research through a thorough investigation to get the relevant data sources and analysis of the results with the help of the selected theoretical framework. To achieve this, the research uses a qualitative approach along with qualitative methods for data collection in order to help answer the proposed research question.

1.3 Research Question

This study seeks to explore the impact of free and open source software on developers' productivity by answering the following question:

1. What is the impact of free and open source software on developers' productivity?

1.4 Delimitation

This study is explored based on the developers' perspectives of FOSS instead of the users' perspectives despite the fact that FOSS can also have an impact on users. Moreover, this study is explored from the perspective of the developers with the aim of understanding what impact FOSS has on developers' productivity at the individual level rather than the organization level. Hence, to explore the research question, developers' self-rated productivity (perceived productivity) measure is used instead of the traditional measures of lines of code or number of commits. While acknowledging that open source libraries are an important component in the free and open source software development paradigm and

equally relevant to the research topic, its usage by the developers and how it impacts their productivity are excluded from this study. Moreover, the reasons why the developers did not share their enhanced or modified versions of the software and its source code was beyond the scope of this research. Hence the research main focus is to look at the impact of free and open source software (FOSS) on productivity through individual developers and information systems lenses.

1.5 Ethical Consideration

Protection of respondents' rights is a fundamental aspect of conducting surveys and interviews. Wohlin (2012) proposed four ethical guidelines with regards to research in the field of information and computing science which are informed consent, scientific value, confidentiality and beneficence. In the process of developing the study, these principles were followed to ensure that the data of the study is relevant and not potentially harmful to individuals. Hence, while collecting respondents' data, the issues of informed consent, anonymity and confidentiality are of paramount importance. The respondents were informed of the purpose of the research and privacy of their data. Before the commence of each interview, verification of consent was taken and the interviewees were made to understand that the interview was going to be recorded, transcribed and their anonymity maintained. Hence, sensitive and confidential information from all the respondents is shared only within the limits stated, even when in conflict with publication interests. The importance of the findings is great enough to warrant the publication of information about the respondents.

1.6 Chapter Disposition

In order to structure the content of this research report in a coherent manner, it has been split into different chapters. Below is a description of the different chapters:

Chapter 2 - Background: The chapter starts with background literature of the key areas relevant to the research topic.

Chapter 3 - Theoretical Framework: This chapter describes the chosen theoretical framework that was leveraged in this research.

Chapter 4 - Methodology: The methodology used in this study to help answer the research question are presented: the research approach, data collection and data analysis methods.

Chapter 5 - Results: This chapter presents the results from the thematic analysis of the data generated from interviews with the help of the theoretical framework. This is followed by a summary of the results.

Chapter 6 - Analysis: This chapter provides a rundown analysis of the themes generated from the interviews' coded data set.

Chapter 7 - Discussion & Conclusion: The research findings are discussed along with the background literature in Chapter 2 and the theoretical framework presented in Chapter 3. Limitations of the study and potential research are also discussed. Finally a summary of the conclusion is made based on key research findings.

2. Background

For the purpose of contextualizing the settings of this research, key areas in which this research topic is built are presented in this chapter. Since this research is about the impact of free and open source software(FOSS) on developers' productivity, it is important to shed light on the evolution of programming languages, FOSS phenomena and its development model. Hence a brief description of the history of programming languages is presented. This is followed by the open source model which encompasses free and open source software development. Thus, key differences between free software and open source software (OSS), the rationality, the pros and cons and language diversity in FOSS development are discussed as well as the quality of FOSS. Software developers who are key players in FOSS development are also discussed. Finally, the concept of productivity, different productivity measures and self-rated productivity (perceived productivity) measure which is used in this research are also discussed.

2.1 History of Programming Languages

Before diving into the open source model which encompasses FOSS development, it is important to build a foundation by understanding the history and evolution of programming languages. Hence, this section will dig deep into the fossils of programming languages and its evolution.

Machine language is the first type of computer language developed by computer scientists. With this language, the computer can only understand the special signals known as binary digits represented by 1‟s and 0‟s respectively as well as hexadecimal characters. This language was difficult to use and prone to errors. Hence to ease the tasks, computer scientist called David Wheeler invented an assembly language also known as "Symbolic Programming Language" because it uses symbols to represent operation code and storage locations (Dasgupta, 2018). Assembly language is a low-level programming language that communicates directly with a computer's hardware. However, assembly language is easier to understand as it was designed to be readable by humans unlike machine languages (Rutishauser et al., 1967; Shih, 2021).

Furthermore, high-level programming languages with compiler implementations were developed between the 1950s and 1960s. These types of languages were commonly called autocode (e.g. COBOL and FORTRAN). However, Fortran which was a machine-independent development of IBM's earlier autocode systems, was the first widespread high-level programming language. Fortran is still a

dominant language for large scale simulations of physical systems and it is currently the 19th most used programming language (García-Rodríguez et al., 2022). Moreover, the Algorithmic programming language called ALGOL is made of automatic programming systems which uses some standard mathematical notation for describing operations (Rutishauser et al., 1967). The ALGOL family is composed of ALGOL 58 and 60 defined between 1958 and 1963 by the EACS (European and American computer scientists), introduced recursion and nested functions under lexical scope. Moreover, ALGOL 60, an extension of ALGOL 58 was identified as the first language with a clear distinction between value parameters, name parameters and their corresponding syntax. ALGOL also introduced various structured programming concepts and its syntax was among the first to be described in formal notation. Some of these structured programming concepts are the “while-do” and “if-then-else” constructs. Around the same period, COBOL also introduced records while LISP introduced a general lambda abstraction in a programming language. However, ALGOL 60 has had a tremendous theoretical and practical impact during the first age of computer science (Dasgupta, 2018). In 1965, ALGOL 68 was developed as a successor of ALGOL 60. According to Dasgupta (2018), the invention of the Assembly language in 1949 to FORTRAN in 1957 and ALGOL programming language between 1958 and 1963 were clear manifestations of the computer scientists’ awareness of incorporating humans into computing experience.

Moreover, these high-level languages refer to the higher level of abstraction from machine languages. Instead of working with registers of the CPU, call stacks and memory addresses, high-level languages deal with objects, variables, arrays, complex expressions, functions, loops, threads and other abstract concepts. The focus is mainly on usability over optimal program efficiency. However, other features such as string handling routines, object-oriented language, and file input/output may also be present. These languages are designed in such a way that the programmers can easily be detached from the machine (Dasgupta, 2018). This is because the power of executing programmers’ instructions has been handed to the machine as the instructions can be executed in the background without the programmers’ knowledge. Moreover, advanced and sophisticated languages like C++ and Java operate at higher levels of abstraction. These usually help designers to separate the concepts that developers need to implement from its instances of implementation. Hence, software developers have been able to focus more on high-level system architectures and less on implementation details (Gu et al., 2018).

In recent years, reusable components or modules and software-container systems play a critical role in innovative software design and development (Rubens, 2017). According to Shih (2021), a strategy that

can be used for accelerating the adoption of new and complex technologies is by increasing the level of abstraction. Abstraction is often useful for understanding innovation and adoption of complex digital technologies, hiding implementation details and reducing the complex interdependencies between components at different levels of abstraction (Gu et al., 2018). Hence, increasing the scope of the innovation through incorporating more of the dependencies raises the level of abstraction. According to Baldwin and Clark (2000), the concept of modularity is key to abstraction. Modularizing a system actually creates a separation of functionality across component or modules (Paparistodimou et al, 2018). Hence, raising the level of abstraction increases software development productivity by reducing the need for extensive understanding of implementation details in return for restricting the degrees of design freedom (Shih, 2021).

2.2 The Open Source Model

The open source development model is the process used by the free and open source community to develop free and open source software (FOSS) projects. It is a decentralized model that is focused on open community collaboration among users and developers who do not belong to typical software development roles (Menéndez-Caravaca et al., 2021). The communities are represented by projects created on the Internet where in the software source code is made publicly available to receive the external contributions of users and developers through voluntary participation (Ferraz et al., 2022). Most of the infrastructure for cloud environments are FOSS developed using the open source model eg. Linux OS projects as well as application servers such as Apache Tomcat.

Furthermore, open source is literally a source code that is made freely accessible for modification, enhancement and redistribution. It also includes permission to use the software source code, libraries source code, design documents etc. with limited protection for intellectual property (Zhizhong et al., 2022). Both software and libraries can be open source as far as they are governed by open source license. While a software is referred to as a collection of instructions that tell a computer how to work, a library is a collection of reusable files, scripts, functions and other resources that can be referenced by programmers mostly during software development (Tsui et al., 2022). However, a free and open source software (FOSS) is a computer software that is released under a copyright license that grants users the right to use, study, enhance, modify, and redistribute the software and its source code to anyone regardless of the purpose. On the other hand, an open source library is any library with an open source license, that indicates that the source code or design can be used, modified, enhanced and/or shared

under defined terms and conditions. Open source libraries are an important component in free and open source software development.

2.2.1 Free and Open Source Software

When you hear the word “free”, one thing that comes to mind is price but “free software” does not actually refer to price but regard for the users' liberty or freedom. Free software is a social movement by the Free Software Foundation (FSF) that promotes both individual and collective control of a software. The movement advocates for the universal freedom to study, redistribute, create, enhance and modify computer software (Bhatt et al., 2016). Moreover, the software evolution is derived by the users and developers through their participation and contribution during software development (Ferraz et al., 2022). The freedom of access is an important aspect for society as it promotes participation and collaboration among users and developers around the globe. For software to be termed as free, it must be made publicly available for both private and commercial use. Since 1983, the Free software movements have been advocating for the freedom and justice of users with the hope of avoiding proprietary software programs that deprive users’ freedom (https://www.gnu.org; Bhatt et al., 2016).

On the contrary, open source software (OSS) grants access to the source code of a software to be studied, enhanced, modified and shared which makes the evolution of the software easy through the community of users and developers. The open source software is usually made available for free in executable form but access to the source code is also provided (Tsui et al., 2022). It can be distributed and freely incorporated into any other software through the open source software licenses. However, there are certain licenses criteria that must be adhered to in terms of access, usage, modification, and distribution of the source code: free redistribution, discrimination, integrity of authors source code, etc. A few examples of open source software are VLC Media Player, Mozilla Firefox, GIMP (GNU Image Manipulation Program) and LibreOffice (https://www.opensource.org).

In summary, free software and open source software describe almost the same range of software but their values and licenses are quite different.

Examples of Free and Open Source Software License

The different available licenses vary in nature and posses key criteria which determine whether the software is termed as free, open source, or proprietary. However, the use of FOSS is governed by FOSS license. This license is a legally valid text that can protect FOSS in terms of use, copy, modification

and distribution (Wang et al., 2021). Some examples of popular software licenses are; GNU General Public License (GPL), The Microsoft Public License (Ms-PL) and the Apache License.

The GNU General Public License is an example of a free, copyleft license for software that grants users the freedom to use, distribute copies, modify and redistribute the software. It allows developers to either publish the modified versions of the software to the public or for private use. Once they decide to publish it, the GPL requires the full source code of the modified version of the software to be made publicly available under the same license (https://www.gnu.org). Hence, GPL is highly restrictive since the published software cannot be relicensed under a different license (Santos, 2017). On the contrary, the Apache is a free permissive license that does not require developers to publish modified versions of the source code but if they choose to then it can be under any license. However, it requires that all original versions of the software be released under the Apache License (Wang et al., 2021). Finally, for Ms-PL license released by Microsoft, the user is not required to publish the source code when the software application is published but if they desire to publish it, then the copyright, trademark originally present in the software must be retained. The terms and conditions of this license is short and precise with a language that is easy to understand (https://www.opensource.org; Papoutsoglou, et al., 2022).

2.2.2 The Rationality in Free and Open Source Software Development

There exists some rationality for every approach to software development. The FOSS concept has brought about a tremendous stir in the arena of software development as it has evolutionized the way individuals and organizations develop, acquire, use, and commercialize software (Tyagi et al., 2022). It encourages open collaboration between developers across the globe during software development. This contemporary approach to software development has drastically evolved over the years. Research shows that startups and commercial projects are increasingly participating in FOSS development. The 10th Annual Future of Open Source Survey showed that 65% of the organisations surveyed have leveraged FOSS to accelerate software development (Anthes, 2016; Taylor & Dantu, 2021).

However, the conventional model has been in existence for long and used by commercial companies for the development of proprietary software. Most proprietary software eg. Microsoft Windows OS with investments worth billions of dollars have gained popularity in the mass market but are now competing with one of the best open source software called Linux OS. The big question is how did this happen?

As opposed to the conventional model, the principal developer of Linux Kernel called Linus Torvalds used a contemporary software development approach by exploring a global pool of talents who develop, test and debug the software released periodically. According to “The Mythical Man Month”, Brooks’ Law states that adding more developers to a software project often resulted in further project delays (Brooks, 1995). However, Stallman (2010), highlighted that Linus Torvalds actually found a workaround Brooks’ Law by making use of more software developers through open access to source code in order to improve software quality. Consequently, vulnerabilities in the software source code can easily be detected since each individual developer looks at the source code from different angles. According to Linus Torvalds, the individual who finds the problem and the one who understands and fixes the problem does not necessarily have to be the same.

However, proprietary software denies users the freedom to take ownership of the software but instead has an owner who controls its access and usage. Hence, it can snoop, track or sometimes compel the users to do harmful upgrades. Proprietary software is a form of combat among citizens because the proprietary owners find ways to sabotage each other rather than prioritizing society's interest and work towards advancement. This can be detrimental to society (Stallman, 2015). According to Vaclav Havel, people need to work for something because it is good and not just because it stands a chance to succeed. Moreover, software access restrictions can limit software adoption rate due to inability to adapt to users’ needs as well as hinder software evolution since developers are trapped to develop from scratch instead of building upon existing software to suit their projects’ needs.

Even though one can argue that the success of FOSS projects is neither automatic nor guaranteed, FOSS has significantly evolved in the last twenty years with an increased adoption rate due to trustworthiness or reliability of some FOSS (Taibi, 2015). As a result, the number of communities is growing and may continue to grow in the upcoming years on account of FOSS becoming a viable alternative to proprietary software (Menéndez-Caravaca et al., 2021). Moreover, FOSS are now considered as good as proprietary software or even better (Li et al., 2022). However, according to Lenarduzzi et al. (2020), the reason that prevents FOSS from gaining a higher adoption rate and being more widespread is the lack of detailed and up-to-date documentation for a large number of projects.

Furthermore, there are varying motivating factors for the adoption of FOSS. However, the motivations for the adoption of FOSS over proprietary software have changed within the last ten years. Results from a study conducted by Taibi (2015), showed that the economical aspects and the freedom of some

type of licenses are not the main factors for adoption of FOSS any more. Instead, other motivations such as the ease of modification and enhancement as well as ethical reasons are considered most important. However, quality, economic and personal productivity are considered of middle importance.

2.2.3 Pros and Cons of Free and Open Source Software

The common attitude towards FOSS is that it saves time and resources since it prevents developers from reinventing the wheel. Majority of the researchers who studied this topic agreed with the statement by Raymond (2001) “given enough eyeballs all bugs are shallow”. This is because the more software developers looking at the source code, the easier it is to find bugs or other vulnerabilities. Moreover, when bugs are reported, dozens of software developers fix it within hours which makes this model more reliable as the software is battle-tested.

In contrast, Steve McConnell (1999), believes that the FOSS development model is an effective model but some resources are not countered because the total effort and the double work of more than one programmer developing the same software makes it inefficient. Moreover, a big challenge and a worthy concern with FOSS is vulnerability that is due to malicious code that may be hidden in a back door or a bad update in the source code (Duan et al., 2019). Hence, McConnell continued to break the principle of “all bugs are shallow” by Raymond (2001). He based his claim on the fact that FOSS projects are not initiated conventionally by initial design and requirements which makes it hard or impossible to easily detect bugs hence, not all bugs are shallow.

Furthermore, Wilson (1999) pointed out an important question with regards to developers skipping the design step in order to respond to market pressure. Since it is not a factor for the FOSS community, why do the adopters keep turning a blind eye on the process that has been proved to make software programming more effective? In response to this question, McConnell (1999) stated and I quote “this is an unusually rapid and iterative form of Barry Boehm’s famous spiral model of software development”. McConnell believed in the loops that iterate over the requirements gathering, design, implementation, as well as testing the bugs and generating new fixes within hours which makes FOSS reliable. He further went on to revise himself on the total effort and double work point that was raised earlier on by stating that it is not necessarily a bad thing as it tends to create a micro competition that will result in initiating the best effort.

According to John Ousterhout, there is a complement between free and open source development and commercial development. This is based on the notion that free and open source projects provide the raw materials that can be further refined and enhanced by commercial developers. Moreover, turning users into co-developers has been proved to speed the debugging process, improve the quality, and acquire new features that may be very important to a wider audience. However, the methodology or the development process is presently so loosely defined that it can hardly be called a “methodology.” (McConnell, 1999; O’Reilly, 1999; Wilson, 1999).

2.2.4 Language Diversity in Free and Open Source Software

Language diversity is relatively sparse in research literature. Based on search results on GitHub which is the most common open source repository hosting site, more than 20 different programming languages have been frequently used in various project repositories. However, Javascript and python were found to be the most used programming languages for different projects while shell and CSS were least used (https://www.github.coms).

Software applications are increasingly developed using multiple programming languages. In a traditional sense, using multiple languages in a single project is uncommon. Most developers often rely on different programming languages to implement different functionalities in a single project. According to a study by Bissyandé et al. (2013), two programming languages are “interoperable” if they are used in the same project. It was also observed that all languages interoperate with one another, though at varying degrees of interoperability. In their findings, shell which is a scripting language that is used in many projects to write automation scripts for compilation, was found to be the language that interoperates the most with other languages.

However, according to a study of linguistic diversity in open source by Vasilescu et al. (2013), there are risks associated with knowledge of programming languages in FOSS communities around multi-language software. One of the risks is having multi-language software projects without enough programmers with knowledge of more than one programming language. Hence, they investigate the readiness of the developer community to take over code implemented in a certain language, and evaluate the risk of not finding contributors that can program using that language. The model used in their study assumes that developers possess more than one programming language skills and these languages are related. Hence the authors mine patterns of shared knowledge of programming languages from developers participating in StackOverflow, a popular programming Q&A website. They base the

mutual intelligibility measure on shared knowledge of the programming languages, as reflected in StackOverflow tags representing programming languages. For answering questions user tags, frequent pairs of user tags indicated that developers possess more than one programming language. While frequent pairs of question tags (e.g., javascript–jquery, asp.net–c#) indicated that these languages are commonly used together.

2.2.5 Free and Open Source Software Quality

The evolution of software is necessary for the success of free and open source software. One of the key concepts of software evolution is that the quality of a software declines as it evolves (Alenezi, 2021). A study on the conflict between open source supporters and opponents regarding the quality of the software developed with open source model was conducted by Stamelos et al., (2002). The opponent team that was not convinced with the idea of FOSS criticized the software development process as undefined. However, according to the supporters, the strength of this model is that the development process is similar to the spiral model with neither risk assessment nor measurable KPIs. The authors highlighted that a key requirement for open source code is that it should be modular, self-contained and self-explanatory in order to easily facilitate remote development.

In the case study by Stamelos et al. (2002), Telelogic Logiscope Code Checker and Viewer functions were used. The tool evaluates components against four criteria: testability, simplicity, readability, and self-descriptiveness. Moreover, the tool uses specific software quality metrics to provide these recommended improvements on the source code: ACCEPT, COMMENT, INSPECT, TEST, REWRITE. A random sample of 100 C Programs in the SUSE Linux 6.0 release were used and all of these are considered applications that can be utilized by Linux systems. The authors described the results as worrying because the source code checker recommended that about 5%–6% of the components need to be REWRITE and 50% of the components of each examined application did not receive the ACCEPT recommendation. Unfortunately, this indicates that the quality of the source code developed for Linux applications failed to meet the requirements of the considered quality standard.

Furthermore, Algic and Höst (2010) from Lund University assessed the changes that have been made by the free and open source community to the Ingres Database management system in terms of software quality. A comparison of two released versions of the software were made: the 2004 version and the 2008 open source version. The following metrics were used: File Function Count (FFC), Line

of Code (LOC), Effective Line of Code (ELOC), Comments Line (C), and The McCab's metric of the Total Cyclomatic Complexity (TCC).

According to the authors, the results of the comparison of source code quality metrics between the files from 2004 version and 2008 version indicate a significant and large decrease in ACC and AELOC. What this means is that the average cyclomatic complexity(ACC) and the average effective lines of code(AELOC) per function have increased somehow for the changed source code. The comparison between the source code quality decrease in metrics is less than the increase of the changed files. Hence, the source code quality metrics for the 2008 version are higher than those of 2004 version. Therefore, the study proved that there is a significant and large increase in quality metrics for source code developed by the free and open source community (Algic & Höst, 2010).

Based on the above scenario, it is quite obvious that the free and open source community should seriously consider the need to develop higher quality software. According to Stamelos, the reason behind the quality problem is that each software programmer develops based on their own coding style and fails to adhere to the SOLID design principles for OOP and thus making it impossible to comply with any coding standard. The research about FOSS quality is still in its baby steps, hence more empirical research is needed to support any claims. However, both above researchers actually recommended that free and open source define its own quality standards and each open source project can have custom source code quality requirements.

2.3 Software Developers

A software developer who is sometimes referred to as a computer programmer, software engineer, programmer or coder, is an individual who develops computer software. Software developers possess technical skills to develop software programs or to oversee their creation by a team and thus, they are the creative minds behind software programs. These software developers build software for private use, commercial use or contribute to FOSS development projects within communities. However, FOSS developers are self- selected, highly motivated or professional developers. According to Smirnova et al. (2022), FOSS developers self-select into FOSS projects within communities where they wish to volunteer their effort.

Free and open source software communities are global, although North American and European countries are over-represented. These communities comprise of core developers who create 80 percent

of the code and the periphery developers who contribute fewer lines of code in projects. According to the general volunteering literature, the periphery developers can be classified as episodic volunteers or contributors. Episodic volunteering describes short-term or infrequent contributions. However, it is possible to retain episodic contributors. Retention of episodic contributors is vital because returning volunteers require less training and retention is one measure of stability in FOSS projects (Barcomb et al., 2022). According to the volunteering literature, satisfaction with the previous volunteering experience, intention to return, and availability are factors that lead to retention (Hyde et al., 2016). However, it was found that higher quality email responses encouraged retention among newcomers within free and open source communities (Steinmacher et al., 2013; Barcomb et al., 2022). Moreover, Smirnova et al. (2022) found that repository owners are able to motivate and retain their talented contributors by showing generosity in accepting external contributors, showing no visible commercial orientation while managing their projects and providing fast feedback.

Furthermore, a recent study conducted through geolocating more than half a million active contributors to GitHub in early 2021 showed a significant increase within Asia, Latin America and Eastern Europe, suggesting a more even spread of FOSS developers globally (Wachs et al., 2022). However, FOSS contributors have witnessed discrimination against non-native English speakers and women (Rodríguez- Pérez et al., 2021). According to Canedo et al. (2019), women are underrepresented in the free and open source communities with less than 10% of the total developers. In a global software developers' survey in 2021, 91.7 % of all respondents were male developers which shows the male-dominating reality of software development careers (https://www.statista.com).

Furthermore, there are various studies on the motivations of free and open source software community developers. Hence various motivating factors have emerged. These motives are usually characterized as intrinsic motives like altruism, enjoyment etc and extrinsic motives like reputation, career and monetary reward (Taylor & Dantu, 2021). Peripheral developers tend to have the same set of motivations as core developers, but developers with extrinsic motives are less likely to continue to contribute (Krishnamurthy et al., 2016; Lee et al., 2017). In particular, extrinsic motives are more common among peripheral developers because they are more likely to seek out opportunities which afford them greater recognition and the chance to gain personal reputation within the communities (Krishnamurthy et al., 2016). Moreover, programming is all about problem solving and problem solving is a skill that is acquired through practice. Therefore, an important question to ask is what

other opportunities are out there that are better than the open source community for software developers to improve their problem solving skills?.

2.4 Productivity

Productivity as a key determinant of success depends on good concentration, technical competence, a responsive environment and a good sense of well-being. The concept of productivity has been in existence for over two centuries and applied in many different circumstances. Although the concept is a widely discussed subject, defining it has proved to be difficult and thus it is often vaguely defined and poorly understood. Moreover, the productivity literature and its various applications revealed that there is neither a unique purpose nor a single definition of productivity (Giovannini, & Nezu, 2001; Umar et al., 2021). Hence, it is important that the basic features that characterize productivity are understood in industry as well as in academia. Otherwise, the existing misinterpretations on the subject will continue to cause issues and declines in productivity. A key thing to note is that productivity is a relative concept, which can not be said to neither increase nor decrease unless a comparison is made. The comparison can be either differences from competitors or other standards at a certain point in time, or changes over time (Tangen, 2002).

Moreover, since there exist several types of productivity, it is also important to understand its ambiguous nature. Hence, being able to separate partial productivity (output related to one type of input) from total factor productivity (output related to multiple types of input) is an important factor. According to Coelli et.al (2005), when the term productivity is used, it is actually referring to a total factor productivity which is a productivity measure capturing all factors contributing to output growth.

Productivity in the field of software development is notoriously challenging to measure due to the complexities of the tasks and processes involved. The traditional definition of productivity: the ratio of output to input use (Giovannini & Nezu, 2001) may sound straightforward, but defining what constitutes input and output in a software development process presents many challenges. The output needs to be evaluated in terms of both quantity and quality, among other dimensions. Regarding the input, the key ingredient in a software development process is people and thus, the qualities and skills of people are also famously difficult to quantify (Wagner & Ruhe, 2018)

Tangen (2002), in his research paper, raised some very important questions that emphasized the need for a basic understanding of the subject of productivity. These questions were “How can we decide

what productivity measures to use if we do not fully understand what productivity is? How can we interpret these productivity measures correctly? How can we know what actions to take in order to improve productivity?". Apparently, the confusion surrounding the concept of productivity poses the necessity to further investigate and emphasize the basic meaning of productivity. Hence, an improper definition of productivity will often result in a misdirected action. Empirical investigations from various studies shows that there is no consensus on how to define productivity(Umar et al., 2021). However, a standard productivity definition is beneficial for measuring productivity.

Hence, a standard definition of productivity commonly used in literature is; Productivity is the ratio of output(s) that is produced by the individuals to the input(s) that was used to produce it.

2.4.1 Productivity Measures

Productivity is commonly defined as a ratio of output to input use. While there is no disagreement on this general notion, the productivity literature and its various applications revealed that there is neither a unique definition nor a single measure of productivity (Giovannini, & Nezu, 2001; Umar et al., 2021). Measuring productivity can be difficult but using the wrong metrics makes it impossible. However some of these productivity measures have changed over the years. According to research, the best way to truly know the productivity level of an individual is to track it using metrics. However, unlike metrics that track sales figures and customer turnover, productivity metrics are used to track and measure how efficient teams or individuals are in completing their tasks and achieving target goals.

Measuring developers' productivity is quite challenging, and it is most likely no single measure will do (Meyer et. al., 2014). According to Wagner & Ruhe (2018), there seems to be a consensus that there are no effective and practical methods to measure knowledge workers' productivity. Even though this problem affects researchers, they have made progress on the issue by developing a breadth of techniques for measuring productivity (Murphy-Hill et al., 2019). Hence, a taxonomy of techniques used in research literature to measure knowledge workers' productivity was developed (Ramírez and Nembhard, 2004; Murphy-Hill et al., 2019). These techniques include function point analysis, self-ratings, peer evaluations, output-input ratio, completion goals etc. These techniques can largely be divided into objective measures like lines of code written per week and subjective measures like self-ratings and peer evaluations (Murphy-Hill et al., 2019). These objective measures have been used as measures of productivity in Software Engineering.

However, Antikainen and Lönnqvist (2006), highlighted that traditional productivity measures are also termed as objective measures, provide an impartial view that is not biased by the individual's personal views and feelings. Moreover, these measures are based on measuring the quantities of output produced and the input used in its production process. Therefore, in organizations where individuals' output can be quantified, it actually makes sense that objective measures are used. However, these objective measures are less applicable to knowledge work like Software Engineering because of the qualitative nature of the output (Prasad et al., 2002; Palvalin, 2019). For instance, knowledge workers like software engineers, developers, architects and scientists who are experts within their field and finding all-encapsulating, objective productivity measures is a strenuous task for these individuals. Moreover, the lines of code metrics can lead to busy work and does not necessarily result in a better product, while the number of commits KPI can result in small and frequent commits that do not move the needle. Hence due to the obstacles in achieving objective measures, subjective measures are widely used.

2.4.2 Self-Rated Productivity

The first step is to define how the study tends to measure productivity. Productivity can be measured on various levels: individual, team-specific and organization. However, in this study, the focus will be on the individual levels of productivity. Hence, self-rated productivity(perceived productivity) which is the most common subjective measure will be used to measure developers' productivity. This productivity measure helps to attain an individual's perspective of productivity. Hence, the way a developer perceives their own productivity is very vital in relation to more objective measures of productivity.

In knowledge work, an objective measure of workers' productivity is often not feasible, and researchers have to rely on self-rated productivity measure. An advantage of subjective measures is that they may capture certain aspects of the input, such as motivation, effort and perceived hindrances to efficient working which would be difficult to quantify objectively. Even though subjective measures may not have the empirical or quantitative appeal of objective measures. However, it is often argued that subjective measures are most appropriate since individuals are likely to work in accordance with their feelings (Moray et al., 1979).

A straightforward and commonly used method to subjectively measure developers' productivity is through interviews or surveys (Ramírez & Nembhard, 2004). Furthermore, using self-report questionnaires or conducting interviews provide a wide view of productivity from different perspectives. Even though they are widely used, it is difficult to validate self-report questionnaires or

interviews because no uniform measure of productivity actually exists (Dunnette & Hough, 1991). However, an advantage of using these techniques is that measuring productivity based on completion of self-determined or project goals has good construct validity. According to research, completion of tasks or accomplishment of projects' goals are the main reasons that software developers report being productive (Meyer et al., 2014). However, neither objective nor subjective productivity measures are universally preferable as they both have limitations. Objective measures tend to suffer from inflexibility. Despite reliability being one of the limitations in using subjective measures, research shows that there is a high level of correlation between subjective measures of productivity and more objective measures of productivity.

3. Theoretical Framework

This chapter focuses on the theoretical framework used in this study. The theoretical framework is important in this research as it helps to clarify the implicit theoretical concept in a manner that is more clearly defined and limits the scope of the relevant data by focusing on specific elements.

3.1 The Feller and Fitzgerald framework

As this study seeks to investigate the impact of free and open source software on productivity of developers, it is important to consider the free and open source software paradigm as these can have direct impact on developers' productivity. For these reasons, the study used a theoretical framework by Feller and Fitzgerald (2000) to provide help in the formulation of the interview questions and guide the data analysis. The Feller and Fitzgerald framework is a resulting framework that was derived from Zachman's IS architecture (ISA) and Checkland's CATWOE framework from Soft Systems Methodology (SSM) which are two previous frameworks that have been very influential in the IS field (Checkland, 1981; Zachman,1987).

The motivation for choosing this resulting framework as lenses for my research is that the Zachman and Checkland's CATWOE frameworks complement each other. Zachman's framework is strong in the area of functions and processes and has different categories for "how" and "when" issues, which do not appear explicitly in the Checkland's CATWOE framework. However, CATWOE's framework has a greater emphasis on people, perhaps it seems unsurprising given its origins. Hence, Zachman's "who" category is significantly elaborated in the CATWOE elements of "client", "actor", and "owner". In terms of the remaining categories , it can be argued that there is quite a close correlation between the elements. Also, CATWOE's element of "environment" aligns with Zachman's "where" and "when" categories. Similarly, "transformation" and "what" can be paired, as can "weltanschauung" and "why".

Based on these, the resulting framework is categorized into the following five constructs or categories: **What** (Transformation); **Why** (Weltanshauung, or World View); **When** and **Where** (Environment); **How**; and **Who** (Client, Actor, Owner). These constructs are briefly discussed in Table 1 below.

What (Transformation)	
What defines a software project as FOSS?	Free and open source software is strictly defined by the license under which it is distributed i.e compliance with the Open Source Definition. However, as with any other emerging concept, there is some fluidity. Hence FOSS is further characterized by some of the dynamics described in this study.
What types of projects tend to be FOSS?	Free and open source software has in the past been dominated by operating and networking systems software, utilities, development tools, and infrastructural components. In recent years an increasing number of productivity and entertainment applications products are being developed.
Why (Weltanshauung, or World View)	
Why are FOSS chosen?	Some of the reasons why FOSS are chosen by many is because of access to open source code, free or no license cost, stability and high level of security.
What are the social motivations for FOSS development?	“Human” motivations for FOSS development include a developer’s desire for advancement through mentorship, peer reputation, the desire for improving programming skills, “meaningful” work, and community oriented idealism.
When and Where (Environment)	
What are the temporal dimensions of FOSS development?	Free and open source software is characterized by the rapid development and rapid evolution of software, frequent and incremental release, and by interaction in “Internet time.”
What are the spatial / geographic dimensions of FOSS development?	Free and open source software is characterized by distributed developer teams, bounded by “cyberspace” rather than physical geography.

How	
How is the FOSS development process organized?	The FOSS model is massive parallel development and debugging. This has traditionally involved loosely-centralized, cooperative, and free contribution from self-motivated individual developers (although there is a recent increase in paid, coordinated development). Free and open source software developers are reckoned to be the most-talented and highly motivated 5% of software developers (Raymond, 1998a).
What tools are used to support the FOSS model?	Massive parallel software development methods are reinforced by the Internet as a communication, collaboration, and distribution platform. Other support tools include academic, non-profit, and commercial patrons of FOSS projects.
Who (Client, Actor, Owner)	
Who are the end users of FOSS products and what are the characteristics of those individuals?	Free and open source software users have primarily been expert users and early adopters. Traditionally there has been a significant overlap between the developers and end users' pool. As more FOSS projects focus on usability and interface issues, this profile will change.
Who are the individual developers of FOSS products and what are the characteristics of those individuals contributing to FOSS projects?	Free and open source software developers have traditionally been self-identified hackers (not "crackers"), professional developers (not amateurs), self-selected, and highly-motivated. Moreover, given the reputation-based culture, developers tend to be (publicly) modest and self-deprecating.

Table 1: A Framework for Analyzing the FOSS development model

4. Methodology

This chapter focuses on the research approach used in the study: qualitative approach, qualitative data collection method, sampling, design choices, method of analysis, and the justification of the design choices. For the purpose of investigating the impact of free and open source software on developers' productivity, qualitative research was used.

4.1 Research Approach

An exploratory qualitative research approach widely used by researchers from the social sciences was used in this study to explore the research topic. In terms or reasoning, an inductive research approach was used to help answer the research question. In an inductive approach, the researcher collects data and creates general assumptions and conclusions based on the analysis of the data i.e from the more specific to the general concept (Jebb et al., 2017). According to Denscombe (2010), there is no single research approach that is regarded as the best in every situation. Selecting the appropriate research approach should be based on research questions and objectives, existing knowledge on the subject area to be researched, time and available resources. The research approach used provides the overall direction of the research and how the researcher will go about answering the research questions (Saunders et al., 2009). Therefore, the approach used in this study is determined by the most appropriate to answer the research question and achieve research objectives.

4.2 Qualitative Research Method

Qualitative research method is commonly defined as "the collection, analysis, and interpretation of interview data, surveys, participant observation, and secondary data in order to understand and describe meanings, relationships and patterns". Qualitative methods provide a unique depth of understanding problems, concepts and experiences. Hence, researchers often use qualitative methods in order to gain a deeper understanding of the human perspective and behavior in various relational and virtual contexts. This method is subjective in nature as it gives an insight into individual perceptions or experience as opposed to the quantitative research method which is objective in nature. Qualitative researchers collect in-depth data from a small focused group in order to obtain meaningful results (Tracy, 2013). These qualitative research findings are often not generalizable, but they provide in-depth description and context of a case.

Hence, to address the research question, interviews were used as a data collection method. This method provides an in-depth understanding of the subject as well as guides the analysis process and thus strengthens the results so that conclusions can be drawn based on solid grounds (Oates, 2005).

4.3 Sampling

Purposeful sampling was done to select the respondents of the interviews. Purposeful sampling involves seeking out respondents who are likely to provide the researcher with rich and informative answers about the research questions under review (Morris, 2015). Hence, software developers familiar with FOSS were reached out through LinkedIn and Whatsapp for participation in the interviews.

The interviews were conducted through online zoom meetings. The invitations to participate in the interviews were sent via Whatsapp and LinkedIn, containing a description of the research study and its purpose. Hence, a follow up zoom link was sent to respondents that acknowledged to taking part in the interviews.

4.3.1 Interviews

Interviews are termed as effective methods of data collection in qualitative research. There are different interview methods that are used in qualitative research: unstructured, semi-structured, structured etc. However, semi-structured interviews offer a more flexible approach to the interview process. Hence the study used semi-structured interviews in order to allow for unanticipated responses and issues to surface through the use of open-ended questions. Although open-ended questions do not provide direct answers to the research question but will give a holistic view of understanding the problem.

Semi-structured Interview Design

A semi-structured interview made up of 10 open-ended questions was conducted and can be seen in full in Appendix A. In line with the more exploratory approach, the questions have been formulated in such a way that it allows some adaptation of new findings from the respondents while remaining very concise and relevant to the research topic. The interview guide was structured into three sections: Demographics, Open Source Software and Perceived Productivity. These different sections are detailed further below.

Demographics: The initial questions were used to gain a background understanding of the interviewees. Questions relating to demographic information such as nationality, age, gender,

profession and employment status were included. These data were only collected as part of the baseline interview which helps profile the interviewees.

Open Source Software: This section consisted of questions that were in relation to the modification or enhancement of FOSS for software development projects by individual developers. It was also meant to understand the factors that influenced developers to use free and open source software development approaches in their projects and motivations in contributing to free and open source community projects. Moreover, the questions were meant to understand the perceived developers' experience in terms of FOSS development model.

Perceived Productivity: This section consisted of questions that were in relation to perceived productivity based on the modification or enhancement of FOSS for software development projects by individual developers. These questions were meant to understand the perceived developers' experience in terms of FOSS development model and if it has an impact on their productivity. This section actually helped in understanding the impact of FOSS on developers' productivity.

Respondents

The results from the interviews represent the responses from the 7 interviewees. These 7 respondents were male developers with over 5 years of experience working in the field across different parts of the globe.

The interviews were conducted online via zoom which allowed the interviewer to reach out to respondents that were not physically available. The interview was carried out over the course of 2 weeks. Invitations to participate were sent out over the period of April as the interviews were conducted simultaneously. The table below displays the interviews that were conducted.

Respondents No.	Profession & Studies	Date	Duration
RI	Software Developer & Masters Student in Data Science at Ca' Foscari University of Venice	12th April, 2022	21 mins
R2	Freelance Software Engineer	13th April, 2022	24 mins
R3	Software Engineer & Masters Student in	15th April, 2022	24 mins

	Engineering Physics at KTH University, Sweden		
R4	Software Developer at Medical Research Council Unit The Gambia at the London School of Hygiene & Tropical Medicine	18th April, 2022	21 mins
R5	Software Developer & Masters Student in Information Systems at Lund University, Sweden	18th April, 2022	22 mins
R6	Senior Research Engineer at Huawei Technologies Research & Development, Edinburgh	19th April, 2022	25 mins
R7	Software Developer & Masters Student in Computer Science at Maharishi International University, USA	23rd April, 2022	21 mins

Table 2: List of Respondents

4.4 Thematic Analysis

This study used thematic analysis through an inductive approach to analyze data from the semi-structured interviews. Thematic analysis is a method for analyzing qualitative data that entails searching across a data set to identify, analyze, and report recurrent patterns (Braun & Clarke, 2006). It is a powerful and flexible method for describing a set of data, but it also involves interpretation in the processes of selecting code and generating themes. Themes which can be generated inductively or deductively are actively constructed meanings derived from a data set that answers a research question.

Thematic analysis is an appropriate and powerful analysis method to use when seeking to understand experiences, thoughts, or behaviors across a set of data (Braun and Clarke 2012). Kiger and Varpio (2020) argued that the choice of using thematic analysis should be based on the aims and objectives of the research, rather than a desire to select an easy to follow method of analysis. Since the method is designed to search for common or shared meanings, it is less suitable for examining unique meanings or experiences from a single person or data item.

According to Varpio et al. (2017), themes do not simply emerge from the data but are constructed by the researcher through analyzing, combining and comparing. In a deductive analysis, predefined theories and/or theoretical frameworks inform theme development, so these themes are often centered on a particular aspect of the data set or a specific question of interest (Braun and Clarke 2006). However, in an inductive analysis, researchers derive themes from the coded data, so the identified themes will be reflective of the entire data set (Braun and Clarke 2006). Hence, the process of coding the data without trying to fit it into a pre-existing conception becomes data driven. However, it is important to note that researchers cannot easily free themselves from their theoretical and epistemological commitments.

The steps that are involved in thematic analysis reflect those of grounded theory, ethnography, and other qualitative methodologies that also rely on coding and searching data sets for themes as part of their processes. Hence this study uses the thematic analysis method outlined by Braun and Clarke (2006), which consists of six steps. With this method, the process of coding the data set and generating the themes are designed to be recursive rather than linear. These six steps are: Familiarizing yourself with the data, Generating initial code, Searching for themes, Reviewing themes, Defining and naming themes, and Producing the report/manuscript.

5. Results

This chapter presents the results from the thematic analysis of data from the interviews. This was carried out through coding of the data and extracting relevant themes. A summary of the results are also presented.

5.1 Results of the Interviews

The table below shows the results of the thematic analysis of the data generated from the interviews. The thematic analysis was done in an inductive approach. Hence, the themes have been developed based on the coded data set from the interviews. The ‘What’, ‘Why’, ‘Who’, ‘When’ and ‘Where’ in the Feller and Fitzgerald framework also helped in the development of the themes.

Coded Data Set	Themes
• Free software is a software that is available for free use but the source code may or may not be available for free • Open source software is a software that the source code are freely available for developers to simply learn, modify and contribute towards its enhancement • Free and open source is a good concept and the best best policy when it comes to software development • The free and open source development paradigm allows one to push technology forward, instead of just keep reinventing the wheel • Low or no license fees • The stability aspect • Productivity, Better functionality, Reliability • Collaboration and the community support is important	**Adoption of FOSS**

• It speeds up the setup process hence saves time and resources • FOSS is useful for learning how the software is developed by the community • The choice of FOSS depends on the type of application one is trying to build • Reliability of FOSS is like a double-edged sword because sometimes one might not get help on time • Security challenges and risks that come with FOSS create room for lack of trust when it comes to adoption	
• Enhancing programming skills • Knowledge Sharing • Gain more experience in the field • Earning money • Long term usage of FOSS products for free	**Motivation for Contribution**
• Poor documentation and lack of documentation of FOSS projects • Risks and security challenges that come with FOSS: not using the most stable version, version upgrade issues, malicious code base etc. • Operating system crashes during modification or enhancement of software	**Encountered Challenges**
• Productivity is the ability to do things quickly and efficiently	**Perceived Productivity**

• Being productive as a developer is to be able to use minimum time to create a software that is of maximum value • Perceived productivity measured in terms of expected output and the time spent in producing that output • Developing software that is useful for the market • Being effective as a team • FOSS saves time and resources • The collaboration actually helps the software to be standardized making development easier • The community provides help and solutions for free use	

Table 3: Thematic Analysis of the interviews

5.1.1 Summary of Interview Results

- All the respondents were male between the ages of 27-35 with over 5 years of experience in the field of Software development.
- All the respondents have a good understanding of FOSS and its development model, hence they all adopted FOSS for their work, university or personal projects
- The main criteria for adoption of FOSS were the open access to source code and low or no license cost of acquiring the FOSS. However, increased productivity, better functionality and stability aspects were also found to be important criteria for adoption.
- While all the respondents acknowledge that they trust FOSS, some had few reservations based on some security concerns. However, there is complete trust when the FOSS is from Apache or trusted free and open source foundations.
- A few of the respondents contributed to free and open source software community projects or participated in forums (StackOverflow).

- The main motivations for contributing to free and open source community projects were to earn money, gain more experience and enhance programming skills. However, knowledge sharing was found to be a motivating factor for participating in forums (StackOverflow).
- The risks and security challenges that come with FOSS like using unstable versions, bad upgrades or malicious software packages create room for lack of trust.
- Poor documentation and lack of documentation were major challenges encountered especially with small FOSS projects. This actually hinders their productivity as developers.
- All the respondents had a common understanding of productivity. However, how each of them perceived productivity as developers slightly vary but fall under one umbrella.
- All the respondents term themselves as productivity based on project goals achieved within minimum time.
- All the respondents are more productive when modifying or enhancing FOSS to suit their project needs with an average overall productivity of 90%. Hence developing the software projects from scratch will consume more time and resources
- All the respondents rated the quality of their modified or enhanced completed version of FOSS as either “Good” or an average of 8 0ut of 10 based on project goals achieved.

6. Analysis

This chapter presents a rundown analysis of themes that have been extracted from the interviews' coded data set .

6.1 Analysis of Interview Themes

The generated themes (See Table 3 on chapter 5 above) were analyzed below.

Adoption of FOSS

All the respondents had a good understanding of what FOSS is and its development model since they have been using it for years. When new technologies emerge, they often face challenges in adoption (Anderson & Tushman, 1990) as the target users have to first recognize their usefulness (value) and then grasp them. Usefulness indicates the 'value' of a software ascribed to it by users. Hence the respondents tried to differentiate the concept of free software from open source software which indicates their understanding of the phenomena. Free software is a software that is available for free but the source code may not be available for free. However, open source software (OSS) is a software that is freely available for developers to simply learn how the software is developed by the community, modify and contribute towards its enhancement. According to all the respondents, FOSS is a good concept as it allows people around the globe to contribute to the projects which increases its security. While R2, R4, R5, R6 and R7 all stated that their work related and personal software development projects are mainly dependent on FOSS, R1, R3 and R6 have modified or enhanced a FOSS for University related projects as well.

Furthermore, according to R4, the choice of FOSS and programming languages to use depend on the type of applications someone is trying to develop. For instance, someone might use a particular FOSS and PHP language if it's a website for a client, or a simple web application that is not memory or network dense. However, if he is developing a software that is memory dense with lots of multitasking and multi-threading, he usually uses Java and Spring Boot framework. According to R1, he used Ubuntu Operating System(OS) for troubleshooting as well as enhanced other available FOSS within Ubuntu to suit his project needs. Moreover, R6 stated that Ubuntu is a top notch OS which he has been using for a long time along with many available Ubuntu software. He added that during his PHD studies, he has modified some of the code of Apache Hadoop in order to collect metrics and also modified a benchmarking tool called High Bench.

According to R3, open source is the best policy when it comes to software development due to the fact that it kind of allows one to push technology forward, instead of just keep reinventing it. Moreover, R7 also stated that one of his reasons for adopting FOSS was to avoid being stuck on some of the issues that have been avoided or resolved by other developers within the community.

> *"....Growing up as a software developer, I learned to avoid what we call reinventing the wheel. Starting from scratch, would mean I would have to avoid so many mistakes that other developers have already avoided it. I always thought it's best to pick up from where people stopped, and then try to enhance it which makes me a more powerful developer than trying to build from scratch."- R7*

All the respondents stated that an important criteria for adopting FOSS was because of open source code and low or no license fees. Both R4 and R7 further went on to elaborate that coming from a low income country and building software for people with low income, license cost is very important. Moreover, R5 added that he has worked in Agile environments and sometimes companies prefer to use free and open source software because it is cost effective.

> *".....many companies prefer open source solutions at the beginning because it reduces cost. Hence if they need something more sophisticated or they are assured that the market will accept their implementation, then they invest more money." -R5*

While R1, R2 and R4 stated that productivity aspect was a very important criteria for adoption of FOSS, R3 and R7 pointed out stability and better functionality respectively as important criteria. Moreover, R1, R3 and R5 also highlighted that using FOSS instead of starting from scratch actually saves time and resources as it speeds up the setup process. When it comes to reliability, the opponents of free and open source software claim that FOSS is unreliable since the source code is publicly available, potential threats can easily be incorporated (Tiwari, 2011). On the contrary, R1 stated that FOSS are reliable in a sense that you can have a problem and then someone else in the community will help you to solve it immediately or the solution is already there for use. However, he added that the reliability comes with a price because sometimes you will not find the solutions so you have to wait for at least a few days before someone could help or no help at all is given.

Even though all the respondents stated that they trust FOSS more than proprietary software due to the collaboration and openness of the source code, four of them highlighted some few security concerns. R7 pointed out that FOSS are more powerful and have more capabilities than proprietary software

because anybody can actually contribute to the development of the software but it comes with a downside. The downside is that their power could be tweaked in such a way that it would go against the users. R5 further went on to add that when one is in a working context, they should really be aware of some risks of FOSS like not using the most stable or latest version as well as security challenges. Moreover, R3 stated that he sometimes is skeptical if the FOSS package he is using is correct or does not work right. He added that this often happens especially when doing mathematical modeling as he wants things to work correctly but it does not really look the way he wants it. Hence he starts having trust issues with FOSS.

However, according to R6 trust issues arise when the FOSS is developed by a single person because he completely trusts FOSS as long as it is from Apache or one of these trusted free and open source software foundations. He added that he would not have any issues using it because there are a lot of developers that collaborate and work on these software and review the source code together before it's pushed into the main branch. Hence, it would be very difficult for someone to include some malicious code as it can be easily detected.

Based on the above, it is clear that the rate of adoption of FOSS by developers is high among these young and experienced developers due to its low license cost, openness of source code, increased productivity, better functionality and stability. However, there are some reserved concerns based on security challenges and risks that come with FOSS which actually create room for lack of trust. Hence, there is a need for the free and open source community to work on improving the security of FOSS in order to minimize the risk of having a reduction in adoption rate of FOSS.

Motivation for Contribution

According to Sawhney and Prandelli (2000), FOSS communities have been portrayed as completely open and chaotic environments. As a result of their inherent instability, such environments have been considered ineffective for pursuing innovative projects in the present turbulent competitive market. However, among all the respondents, two actually contributed to FOSS development projects in communities. Even though the remaining five respondents had the intention to contribute to these projects, they could not fulfill their wish due to busy schedules. However, three among these five respondents actually participated in free and open source community forums like StackOverflow by opening questions or threads or helping to answer questions and queries from people. Moreover, R4 stated that he has participated in the community forums by helping other people seeking answers to

queries mostly related to Drupal. His main motivation for helping others is that Drupal has a lot of modules which are available for free that as a developer he could use. So he felt obligated to start contributing by helping others. He added that as a developer, you can learn a lot from other developers. So knowledge sharing is very important because if others do not share their knowledge, then everyone will find life very difficult.

On the contrary, R2 has contributed to a FOSS development project for the past two years while working at a National Research Center in Sweden. It was a micro-service architecture with four services that fall under the same project. However, his contribution to the project was based on job related tasks because he rarely participated in forums or contributed to FOSS development projects in open source communities. Hence his main motivation was to earn money.

Furthermore, R6, highlighted that since he has been using Ubuntu for a long time for free, he always had the intention to one day try and contribute to FOSS development projects in communities. So while doing his PHD in 2019, he contributed to the Apache Gora project. This was an open source project that involved building an abstraction on top of different NoSQL databases. Hence users do not have to use NoSQL specific API to interact with those databases but instead can have a unified API that they can use to interact with any of those databases.

> *"...for example, if you have MongoDB, and you have Cassandra, and you have another kind of NoSQL DB. All these databases have different kinds of languages that would be used to interact with each other. Hence, what Apache Gora did is to kind of build an abstraction on top so that users can have a unified API and interact with any of those databases which kind of simplifies the process." - R6*

He also stated that there is a program sponsored by Google called Google Summer of Code GSoC where he spent the summer helping them to develop a benchmarking module. His motivations for contributing in these FOSS projects were to gain more experience and enhance his programming skills as one can learn a lot through collaborating with people with different experiences. He added that another motivation was to earn money. This is because when you contribute to some Google projects, at the end of every cycle, Google gives you money as compensation which you can use as a student to settle some of your needs.

Therefore, It is quite clear that even though all the respondents have adopted FOSS, just a few actually contributed to community projects or participated in forums. Those that contributed or participated

were motivated to gain experience, enhance their programming skills, earn money and share their knowledge.

Encountered Challenges

In software development, it is most likely that you will run into some problems. According to R1, after installing Ubuntu in his machine and then enhancing the FOSS source code to suit his project needs, the Operating System crashed and the user interface changed. However, these changes only affected his machine and not the general community.

Furthermore, one of the challenges faced by R4 was the upgrade of .NET core from the LTS version of version 3 to version 6. He added that it was a challenge to upgrade an existing software application to that version. Moreover, there are risks and security challenges like having a FOSS package with malicious source code and unstable versions.

According to R3 and R6, the biggest problem they encounter with FOSS is poor documentation and lack of documentation with FOSS projects. Hence, the liberty and openness of FOSS comes with a price. As a developer trying to read someone's code that is not well documented, you will have a problem understanding what they're trying to do.

> *“....It's kind of like a double-edged sword. For instance, if you look at proprietary software, it's pretty well documented. They have people working on it during downtime to document what to do and everything you need. However, when it comes to free and open source software, if the project is not a big one, there is usually a lack of documentation and understanding of it. So as a developer, you need to dig into the code and try to understand it which takes some time and hinders my productivity.” - R3*

R3 further went on to give an example of a free software machine learning library for Python called scikit-learn (Sklearn) which is not well documented. He stated that there is a tree drawing function that basically plots a graph of a tree, but the problem with that plotting library is that it plots trees with two branches only, and not multiple branches. However, this aspect was not part of the documentation. So he had a problem where he needed to plot a lot of branches. Hence, to do that, he had to break down the library and modify it which is very time consuming and annoying.

R6 stated that with Apache Gora at least if you run into a problem, you could ping one of your mentors or the contributors and tell them you need help because you do not understand this particular part as its poorly documented or undocumented,

> *"….Apache Gora project has a support base. While working on the project, you're also working with some other contributors that you have direct contact with. There is also a mailing list. You can always send an email to the mailing list and if you are working on that project, they are always eager to see who is emailing and what their queries are. Hence anyone that knows the answers to the queries can obviously reply."- R6*

However, R6 further added that his experience with High Bench was different. While he was trying to modify the software and encountered some issues, he wrote to the developer of the software who works at Intel. He stated that the developer did not reply to his query probably because it is not his problem.

According to R5 and R7, the challenges they encountered were not something that was impossible to figure out themselves. R5 further went on to highlight that the problems were usually trivial issues that he could fix or the community could easily help.

> *" …..It's either the code is not doing what I need, so I tried to fix it or the rendering of this element in HTML is not working in Internet Explorer so I try to use compatibility or something else. Although it can be easily fixed by other people, I prefer to do it myself because I have the pressure of being in the working context to tackle challenges."- R5*

R7 added that the challenges faced were based on his own understanding of how he wanted to modify the existing FOSS to suit his needs. The implementation was there so it is not like the possibility was not there. Hence he did research and some trial and errors to make sure it works towards his needs.

Though some challenges can be based on developers' lack of understanding or design choices, the security issues, poor documentation and lack of documentation challenges encountered are worthy concerns. While these challenges seem to be less in big FOSS projects, the free and open source community should work towards addressing these issues regardless of the size of the projects and thus developers can be more productive.

Perceived Productivity

The productivity literature revealed that there is neither a unique definition nor a single measure of productivity (Giovannini, & Nezu, 2001; Umar et al., 2021). According to Endler and Magnusson (1976), human beings operate on perceptions versus reality, so subjective measures (perceived productivity) are important. However, individual productivity can be perceived differently. All the respondents stated that productivity is important to them personally. While R1 stated that productivity is achieving project goals within a short period of time, R2 and R6 defined productivity as the ability to quickly and efficiently complete projects or tasks. According to R2, his measure of productivity is based on how quickly he is doing his projects' tasks and how efficient it is when completed. Hence the goal is to use minimum time to create a software that is of maximum value.

However, R5 highlighted that to be productive as a developer especially in an Agile environment is to be effective as a team and develop software that is useful for the market. According to R7, he perceives himself as productive when he delivers software solutions that suit the needs of his clients. R4 added that being productive as a developer is to be able to implement your project tasks in the time frame that you allocated for it. So as a developer with over 8 years of experience, he measures his productivity based on expected output and the time spent in producing that output. Hence choosing the right type of FOSS is very important as it determines the projects' outcome.

Furthermore, R1 stated that his projects tasks were usually University related assignments with a duration of at least 3 days for enhancement and submission. However, by enhancing available FOSS software, it usually took him more than a week to complete the tasks because at times it was very difficult to understand what exactly to do. He added that based on the mode of some of his projects' tasks, it might have been difficult to use Windows OS because it is very rare to see people using Windows OS for scripting and using other available Windows scripting software might have taken a longer duration to complete the tasks.

According to R6, starting from scratch was going to be very challenging because he would have to write all the code himself. For instance, it would have been difficult for him to write Apache Hadoop alone because that was the platform he was trying to benchmark. For the benchmarking tool, its code base is quite huge and tons of developers from Intel also worked on it. Hence working on it alone would not be as effective and efficient as several other developers working on it.

Moreover, R3 stated that if the FOSS he modified or enhanced were not available, it would have taken him three or four times the duration of the projects which were between one to six months. Hence a lot of the good things with FOSS is that most of the time you do not really need to have expertise in the area as you can just use or tweak what is already available. R7 stated that while trying to tweak other peoples' ideas to suit his project needs, he went beyond the project duration of four months. So developing the software from scratch would have hindered his productivity as his project goals will be achieved beyond a year.

Since quality is an important factor in software development, it should be considered during software evaluation. According to R5, the quality of his projects were good as it aligned to expectations. Moreover, R4 highlighted that the quality of the modified versions of his software projects were good due to the fact that most of the infrastructure has already been built and tested. R2 added that the collaboration in FOSS development actually helps the software to be standardized, hence yields a good quality. Furthermore, R1 stated that quality was an important factor for the evaluation of his assigned projects' tasks. Hence, he provided a rating of 8 out of 10 for the quality of his submitted projects' tasks because even though the projects were not completed, he was able to achieve at least 90% of his goals. R6 added that he would not say his project was an excellent piece of work but good enough to have created an impression on his mentors. Hence he provided a rating of 8 out of 10..

> *"....so I was given a pass and my mentors were quite impressed with what I've done. So I would not say it' was an excellent piece of work, but I will rate it and give it an 8 out of 10." - R6*

All the respondents stated that they tend to mark their software projects as complete if all the functionalities are implemented and project goals are achieved. R6 added that no matter the number of hours spent working on a project, if the goals of the project are not achieved then the project is incomplete. All the respondents stated that they feel productive almost all the time when modifying or enhancing FOSS to suit their projects' needs. Moreover, the majority respondents gave an average rating of 90% overall productivity in getting their software projects completed. However, one of the respondents gave a 70% rating that was further justified.

> *"....I am 70% productive most of the time. In the cases that I feel like the solution was not aligned to users expectations were not due to bad technical decisions but more of bad product owner decisions or business analysts. For example, I think in only 10% of the cases the*

free and open source solutions that we used was the problem and the other 20% was due to misalignment between the business investors and the development team."- R5

Though all the respondents perceived productivity slightly varies, they all had a common understanding of productivity. Hence, they term themselves as productive based on project goals achieved within minimum time. They are more productive with an average overall productivity of 90% when modifying or enhancing FOSS to suit their project needs. The quality of their modified or enhanced completed version of FOSS is Good. However, developing the software projects from scratch will consume more time and resources and thus lower their productivity level.

7. Discussion & Conclusion

In this chapter the results of the interviews are discussed by leveraging the chosen theoretical framework and background literature. Hence the research question “ What is the impact of free and open source software on developers’ productivity” will be answered based on the analysis performed in the previous chapter. This is followed by Research contributions, limitations and future research and summary of conclusion.

7.1. Discussion

Based on the results from the survey and interviews, it is clear that all the respondents have a good understanding of FOSS as well as productivity. All the respondents have adopted FOSS and its development model either for work, University or personal projects as they were employed and / or students. All these respondents were male between the ages of 24-35. Hence it's not surprising that in a global software developer survey in 2021, 91.7% of all respondents were male developers which shows the male-dominating reality of software development careers (https://www.statista.com). Moreover, women are underrepresented in the FOSS communities with less than 10% of the total developers (Canedo et al. 2019).

Furthermore, in accordance with the “Why” in the Feller and Fitzgerald framework, some of the reasons for choosing FOSS is because of access to open source code, free or no license cost, stability and high level of security. According to the results from Taibi (2015), free or no license cost and freedom of access are not the main factors for adoption of FOSS. Instead, ease of modification and enhancement as well as ethical reasons are considered the most important factors. On the contrary, it is clear from the study findings that open access to source code and low or no license cost of acquiring the FOSS were the main criteria for adoption among all respondents. License cost is very important when choosing a software especially in low income countries. This is because if you choose to use software with high license cost like proprietary software, the cost at which you will sell it will be difficult for individuals or small businesses to be able to afford the software. So if it's free and open source software, it totally kicks out the cost of the license. However, stability and security as well as the criteria which were not included as part of the “Why” i.e. increases productivity and better functionality actually emerged as important criteria for adoption among the majority of the respondents.

Though reliability was found to be a less important factor for adoption of FOSS among the respondents, it might have been an important factor for adoption among end user respondents. This could be because the developers perceived reliability to be important in terms of usability and thus more geared towards the users' perspective. However, reliability and stability are critical factors when it comes to FOSS development (Taibi, 2015). In terms of stability, working with unstable versions of FOSS can lead to a lot of issues during the development process. It might require the developers to refactor the whole code base which could consume a lot of resources and time. On the other hand, reliable software facilitates smooth development and upgrade processes and thus leads to increased productivity. Moreover, developers usually require support from the community in case of encountered issues while trying to modify or enhance FOSS to suit their projects' needs. However, the reliability of FOSS is like a double-edged sword due to the fact that sometimes developers fail to get the needed help on time and thus can hinder their productivity.

Moreover, while FOSS are termed unreliable due to open access to source code, it can be argued that "given enough eyeballs all bugs are shallow"(Raymond, 2001). This is because the more developers looking at the code base, the easier it is to find bugs or other vulnerabilities which helps to increase security and stability of the software. However, security challenges emerged as a worthy concern among the respondents. While all the respondents declared trust for FOSS, a few had some reserved percentages for few concerns. The security challenges and risks that come with FOSS like malicious software packages, using unstable versions or bad upgrades actually create room for lack of trust. These are common challenges encountered by most developers while working with FOSS. The openness of the source code actually allows anyone to do whatever they want with the codes. Hence, hackers usually use this transparency in order to look for vulnerabilities to exploit.

On the other hand, Lenarduzzi et al. (2020), highlighted that lack of detailed and up-to-date documentation for a large number of FOSS projects actually hinders the widespread and high adoption rate for FOSS. In accordance with this, the findings from the respondents indicated that the biggest challenges encountered with especially small FOSS projects are poor documentation and lack of documentation. Most of the time the free and open source communities fail to invest time and resources in developing proper documentation for FOSS projects due to it being free. This usually happens with most small FOSS projects. What the communities fail to understand is that developers depend on these software documentations to actually understand the software source code and thus enhance or modify it to suit their needs. With the lack of documentation or improper documentation, developers need to dig

deep into the source code to understand how to proceed which consumes time and resources and thus hinders their productivity. This can also lead to developers' lack of adoption of FOSS.

Even though there was a common understanding of productivity among all the respondents, how each of them perceived productivity as developers slightly varies but falls under the same umbrella. All the respondents perceived themselves as productive as long as they achieved projects' goals with limited resources. Hence the goal is to use minimum time and resources to create software that is valuable. According to Palvalin et al., (2017), the ideal form of improved productivity is to increase the output while decreasing the input. In relation to this, all the respondents were able to achieve their software projects' goals with a decrease in resources used. These respondents stated that they are more productive when modifying or enhancing FOSS to suit their projects' needs as it saves time and resources. Based on the analysis, it could be seen that these respondents were actually able to complete their projects within the actual project duration by modifying or enhancing existing FOSS projects. Overall the quality of their modified or enhanced completed versions of FOSS projects were "Good" and these ratings were based on project goals achieved and not only on hours worked on the projects. Hence developing the projects from scratch was not feasible as it would consume more resources, exceed project duration and lead to reinventing the wheel.

Furthermore, according to the "Who" in Feller and Fitzgerald framework, FOSS developers have traditionally been self-identified hackers, professional developers, and highly-motivated individuals. It could be seen that the respondents were professional developers, self-selected, and highly-motivated individuals from different geographical regions though the majority were from sub-Saharan Africa. Hence, this is in accordance with the Feller and Fitzgerald framework "When" and "Where" which highlighted that FOSS is characterized by distributed developer teams within communities, bounded by "cyberspace" rather than physical geography. Moreover, some of the respondents did not only modify or enhance an existing FOSS for their projects' needs but actually contributed voluntarily to FOSS development projects within communities. Smirnova et al. (2022), stated that FOSS developers self-select into FOSS projects within communities where they volunteer their effort.

Moreover, the "Why" in the Feller and Fitzgerald framework outlined social motivations for developers contributing to FOSS projects in communities. Some of these social motivations such as enhancing programming skills and gaining more experience in the field emerged as very important motivating factors among the respondents. The field of software development is evolving and programming

languages are also evolving, hence the desire to learn new technologies and enhance skills. However, gaining personal reputation and getting job opportunities did not emerge as an important motivating criteria. This might be because the respondents were already employed and satisfied with their jobs. However, gaining personal reputation within the communities might create room for new or better job opportunities. According to the “How” in Feller and Fitzerald framework, there is a recent increase in paid and coordinated development within FOSS communities. Moreover, analysis of 2017 GitHub Open Source Survey data of over 5,500 randomly selected FOSS developers indicated tangible motivations such as the monetary reward are relatively considered more important factors for developers contributing to community projects (Taylor & Dantu, 2021). This highlighted monetary aspect actually emerged as a motivating factor for contributing among the respondents.

7.1.1 Research Contributions

The study presents a comprehensive analysis of the FOSS development model and the impact it has on software developers’ productivity. Hence it creates a new angle of exploring developers’ productivity from a subjective perspective rather than the traditional objective perspective. This will provide grounds for information systems(IS) researchers to shift their focus and consider exploring software developers’ productivity in a more subjective approach. Furthermore, this study also highlighted and discussed some of the challenges faced by developers when using, modifying or enhancing free and open source software. Such discussions will enable other researchers in the IS field to focus on these issues and try to do a thorough investigation on the subject matter. Moreover, this will allow the free and open source software community to find ways of overcoming these challenges faced by developers in order to facilitate a seamless software development process.

7.1.2 Limitations and Future Research

This study used subjective measures of developers’ productivity as opposed to traditional objective measures. However, the author acknowledges that there are limitations in using the subjective measures with the major one being reliability of the data. According to Smeed and Hayter (2010), combining subjective and objective measures may be the most appropriate. If both subjective and objective measures of productivity are taken then this will avoid some of the limitations of using each measure alone and provide a more holistic view of productivity. However, it is with the authors hope that other researchers will use a hybrid of subjective and objective measures of developers’ productivity to dig deeper into this topic.

Moreover, the data collection for the study was gender biased as all the respondents were male. Therefore, the author acknowledges that the topic could have been explored more with a diverse gender base.

According to Nagle (2015), despite the abundance of literature on the supply side, there is almost no literature on the demand side (end users perspective) of FOSS usage and its impact on productivity of these users; who uses it, why do they use it, and what effects does using it have on productivity remain unanswered questions. Hence it would be of great benefit if more researchers try to dive into these topics and provide answers to these unanswered questions.

Based on the analysis of the interviews, it was clear that the majority of the respondents did not adhere to the FOSS license by publicly sharing the modified or enhanced version of the software or source code. This was mainly developers from developing countries within sub-saharan Africa. While the scope of the research did not allow me to dive into this part, further research is needed to determine the reasons why developers within these regions are less likely to publicly share their source code. Is it that the free and open source software license does not fit into the context of sub-saharan Africa?

Licensing is very important in software development as it differentiates free and open source software from proprietary software. However some licenses of both free and open source software have very restrictive policies when it comes to modifying, enhancing or sharing the software or source code. As a result, most developers find difficulties with free access to FOSS or the source code based on license restrictions and thus might limit their contributions for the evolution of the software . Hence, it would be interesting for researchers to explore what kind of licensing model most developers that contribute to FOSS development projects within communities are likely to contribute.

However, the author also acknowledges that the methodology used in this study is just one of many possible ways of exploring this research topic. As a result, the methods used are neither declared as the best nor a complete way of exploring this research topic, but the one to the best of the ability of the author. Hence, it is with the authors hope that other researchers will take it up from here and explore the topic more by touching upon other angles that are relevant to this topic.

7.2 Conclusion

The research aimed to explore the impact of FOSS on developers' productivity based on self-rated productivity measure. The findings indicate that FOSS actually has a positive impact on developers'

productivity and the rate of adoption is high among young and experienced developers. However, the risks and security challenges that come with FOSS based on its open model actually creates room for lack of trust which can reduce the adoption rate. Though FOSS has improved developers' productivity when it comes to software development, the challenges of poor documentation and lack of documentation encountered when trying to modify or enhance FOSS hinders developers' productivity. Hence, this research findings clearly illustrates that FOSS has actually improved developers' productivity but it also raises the question; "what should the free and open source software community do to address these challenges?". Despite the limitations of this study, the approach used, the collected data and the related analysis actually provide a unique qualitative picture of FOSS and its impact on developers' productivity.

References

Alenezi, M. (2021): Internal quality evolution of open-source software systems, College of Computer and Information Sciences, Prince Sultan University, Riyadh, 11586, Saudi Arabia

Algic, A. & Höst, M (2010):A Case Study on the Transformation from Proprietary to Open Source Software, Department of Computer Science, Lund University, Sweden P. ˚Agerfalk et al. (Eds.): OSS 2010, IFIP AICT 319, pp. 367–372.

Anderson, P., & Tushman, M. (1990): Technological discontinuities and dominant designs: A cyclical model of technological change. Administrative Science Quarterly, 604-633

Anthes, G. (2016). Open source software no longer optional. Communications of the ACM. 59. 15-17.

Antikainen, R., & Lönnqvist, A. (2006): Knowledge work productivity assessment.

Baldwin, C., & Clark, K. (2000): Design rules: The power of modularity (Vol. 1). MIT Press.

Barcomb, A., & Stol, K.J., & Fitzgerald, B., & Riehle, D.(2022): Managing Episodic Volunteers in Free/Libre/Open Source Software Communities, in *IEEE Transactions on Software Engineering*, vol. 48, no. 1, pp. 260-277, 1 Jan. 2022, doi: 10.1109/TSE.2020.2985093.

Bhatt, P., & Ahmad, J.A., & Roomi, A.M.(2016): Social innovation with open source software: User engagement and development challenges in India, Technovation, Volumes 52–53,

Bissyandé, T.F., & Thung, F., & Lo, D., & Jiang, L., & Réveillère, L.(2013): Popularity, Interoperability, and Impact of Programming Languages in 100,000 Open Source Projects. 2013 IEEE

Braun, V., & Clarke, V. (2006): Using thematic analysis in psychology. Qual Res Psychol. 3(2):77–101.

Braun, V., & Clarke, V. (2012): Thematic analysis. In: Cooper H, editor. APA handbook of research methods in psychology. Vol. 2, research designs.Washington (DC)

Brooks, F. P. (1995). The mythical man month. Reading, Mass, Addison-Wesley.

Canedo, E.D., & Tives, H.A., & Marioti, M.B., & Fagundes, F., & de Cerqueira J.A.S. (2019): Barriers faced by women in software development projects. Information 10(10):309.

Coelli, J.T., & Prasada Rao, S.D., & O'Donnell,J,C., & Battese, E,G.(2005): An Introduction to Efficiency and Productivity Analysis, Second Edition, Springer.com.

Dasgupta, S. (2018): The Second Age of Computer Science: From Algol Genes to Neural Nets, Oxford University Press.

Duan, R., & Bijlani, A., & Ji, Y., & Alrawi, O.,& Xiong, Y., & Ike, M., & Saltaformaggio, B., & Lee, W. (2019): Automating Patching of Vulnerable Open-Source Software Versions in Application Binaries.

Ejlertsson, G., (2005): Enkäten i praktiken - En handbok i enkätmetodik. Studentlitteratur, Lund

Feller, J., & Fitzgerald, B. (2000): A framework analysis of the open source software development paradigm. 58-69.

Ferraz, I., & Santos J.C. (2022): Transformation Of Free And Open Source Software Development Projects: Governance Between The Cathedral And Bazaar. Revista de Administração de Empresas.

García-Rodríguez, M., & Añel, J.A., & Rodeiro-Iglesias, J. (2022): Assessing and improving the quality of Fortran code in scientific software: FortranAnalyser.

Giovannini, E., & Nezu, R.(2001): Measuring Productivity: Measurement Of Aggregate And Industry-Level Productivity Growth, OECD Manual 2001.

GitHub: https://www.github.com (Accessed May 2022).

GNU Operating System: https://www.gnu.org (Accessed May 2022).

Gu, R., & Shao, Z., & Kim, J., & Wu, X.N., & Koenig, J., & Sjöberg, V., & Chen, H., & Costanzo, D., & Ramananandro, T. (2018): Certified concurrent abstraction layers.

Hyde, M.K., & Dunn, J., & Bax, C., & Chambers, S.K.(2016): Episodic volunteering and retention: An integrated theoretical approach, Nonprofit Voluntary Sector Quart., vol. 45, no. 1, pp. 45–63, 2016.

Jebb,T.A., & Parrigon, S., & Woo, E.S.(2017): Exploratory data analysis as a foundation of inductive research, Human Resource Management Review, Volume 27, Issue 2.

Kiger, M & Varpio, L. (2020): Thematic analysis of qualitative data: AMEE Guide No. 131. Medical Teacher. 42. 1-9. 10.1080/0142159X.2020.1755030.

Krishnamurthy, R., & Jacob, V., & Radhakrishnan, S., & Dogan, K.(2016):Peripheral developer participation in open source projects: An empirical analysis, ACM Trans. Manage. Inf. Syst., vol. 6, no. 4, pp. 14–45, 2016.

Lee, J.A., & Carver, C., & Bosu, A.(2017): Understanding the impressions, motivations, and barriers of one time code contributors to FLOSS projects: a survey, in Proc. 39th Int. Conf. Softw. Eng., 2017.

Lenarduzzi, V., & Taibi, D., & Tosi, D., & Lavazza, L., & Morasca, S. (2020): Open Source Software Evaluation, Selection, and Adoption: a Systematic Literature Review. 10.1109/SEAA51224.2020.00076.

Li, X., & Moreschini, S., & Zhang, Z., & Taibi, D. (2022): Exploring factors and metrics to select open source software components for integration: An empirical study. Journal of Systems and Software. 188.

Mata, J.F., & Quesada, A. (2014): Free and Open Source Software in Costa Rican Local Governments. CLEI Electronic Journal. 17. 6-6. 10.19153/cleiej.17.2.5.

McConnell, S. (1999): Open source methodology: ready for prime time? IEEE Software, 16 (4), 6–8.

Menéndez-Caravaca, E., & Bueno, S., & Gallego, M.D. (2021): Exploring the link between free and open source software and the collaborative economy: A Delphi-based scenario for the year 2025,

Meyer, A.N., & Fritz, T., & Murphy, G.C., & Zimmermann, T. (2014): Software developers' perceptions of productivity, Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2014. p.19-29. DOI: 10.1145/2635868.2635892.

Moray, N., et al. (1979): Final report of the Experimental Psychology Group. Mental Workload: Its Theory and Measurement, ed. Moray, N., 101–114. New York: Plenum

Murphy-Hill E.,& Wagner S.(2019): Software Productivity Through the Lens of Knowledge Work. In: Sadowski C., Zimmermann T.(eds) Rethinking Productivity in Software Engineering. Apress, Berkeley,

Nagle, F. (2015): Crowdsourced Digital Goods and Firm Productivity: Evidence from Open Source Software. SSRN Electronic Journal. 10.2139/ssrn.2559957.

Nagy, D., & Yassin, A.M., & Bhattacherjee, A. (2010): Organizational adoption of open source software: Barriers and remedies. Commun. ACM. 53. 148-151. 10.1145/1666420.1666457.

Oates, B. J.(2005): Researching Information Systems and Computing (1st ed.). SAGE Publications Ltd.

Open Source Initiative: https://www.opensource.org (Accessed May 2022).

O'Reilly, T. (1999) Lessons from open source software development. Communications of the ACM, 42 (4), 33–37.

Palvalin, M.,& van der Voordt, T., & Jylha, T. (2017): The impact of workplaces and self-management practices on the productivity of knowledge workers, Journal of Facilities Management Vol. 15 No. 4, pp. 423-438

Palvalin, M., (2019): What matters for knowledge work productivity?,Employee Relations, Vol. 41 No. 1, pp. 209-227. https://doi.org/10.1108/ER-04-2017-0091

Paparistodimou, G., & Duffy, A., & Knight, P., & Whitfield, R., & Robb, M., & Voong, C. (2018): Network-based metrics for assessment of naval distributed system architectures.

Papoutsoglou, M., & Kapitsaki, G.M & German, D., & Angelis, L.(2022): An analysis of open source software licensing questions in Stack Exchange sites, Journal of Systems and Software, Volume 183.

Prasad, M., & Shih, Y. C. T., & Wahlqvist, P., & Shikiar, R. (2002): PMD19 A critical review of health-related productivity measures.Value in Health, 5(6), 535-536.

Ramírez, Y.W., & Nembhard, D,A.(2004): Measuring knowledge worker productivity: A taxonomy, Journal of Intellectual Capital, vol. 5, no. 4, pp. 602–628, 2004.

Raymond ES. (2001): The Cathedral & The Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. O'Reilly Media: Sebastopol, CA.

Raymond, E. (1999a): The Magic Cauldron,(http://www.tuxedo.org/~esr/writings/magic-cauldron/magic-cauldron.html; accessed May 1, 2000).

Rodríguez-Pérez, G., Nadri, R. & Nagappan, M. Perceived dversity in software engineering: a systematic literature review. *Empir Software Eng* **26,** 102 (2021).

Rubens, P. (2017): What are containers and why do you need them?. CIO. URL: https://www. cio. com/article/2924995/software/what-are-containers-and-why-do-you-needthem. Html.

Rutishauser, H., & Bauer, L.F. (1967): Handbook for Automatic Computation: Description of Algol 60, By Springer

Santos, C.D. (2017): Changes in free and open source software licenses: managerial interventions and variations on project attractiveness. *J Internet Serv Appl* **8,** 11 (2017).

Saunders, M., & Lewis, P., & Thornhill, A. (2009): Research Methods for Business Students. Pearson, New York.

Sawhney M., & Prandelli, E., (2000): Communities of creation: managing distributed innovation in turbulent markets, California Management Review, vol. 42, pp. 24–54.

Smirnova, I., & Reitzig, M., & Alexy, O. (2022): What makes the right OSS contributor tick? Treatments to motivate high-skilled developers, Research Policy, Volume 51, Issue 1, 104368.

Shih, C.W. (2021): Increasing the Level of Abstraction as a Strategy for Accelerating the Adoption of Complex Technologies. Strategy Science. 6. 10.1287/stsc.2020.0116.

Smeed & Hayter.(2010): Well-being at work survey 2010.

Stallman, R.M (2010):Free as in Freedom (2.0): Richard Stallman and the Free Software Revolution, Free Software Foundation.

Stallman, R.M. (2015): Why Open Source Misses the Point OF Free Software. Free Software, Free Society, Selected Essays of Richard M. Stallman Third Edition, Free Software Foundation.

Stamelos, L., Oikonomou, A., & Angelis, L (2002): Code quality analysis in open source software development: Enhancing open source release quality, Department of Informatics, Aristotle University of Thessaloniki, Information Systems Journal 12, 43–60.

Statista: https://www.statista.com(Accessed May 2022).

Steinmacher, I., & Wiese, I., & Chaves, A.P., & Gerosa, M.A.(2013): Why do newcomers abandon open source software projects?, in Proc. 6th Int. Workshop Cooperative Hum. Aspects Softw. Eng, pp. 25–32.

Taibi, D. (2015): An Empirical Investigation on the Motivations for the Adoption of Open Source Software.

Tangen, S. (2002): Understanding the concept of productivity.

Taylor, J., & Dantu, R.(2021): For Love or Money? Examining Reasons behind OSS Developers' Contributions, Information Systems Management.

Tiwari, V. (2011): Reliability Issues in Open Source Software. International Journal of Computer Applications. 34. 34-38. 10.5120/4065-5849.

Tracy, S.J. (2013): Qualitative Research Methods, Collecting Evidence, Crafting Analysis, Communicating Impact.

Tsui, F., & Karam, O., & Bernal, B. (2022): Essentials of Software Engineering, Fifth Edition

Tyagi, S., & Kumar, D., & Kumar, S. (2022): Open source software: analysis of available reliability models keeping security in the forefront. International Journal of Information Technology.

Umar, M., & Ali, S., & Sial, M. (2021): The Impact of Knowledge Governance on Knowledge Workers' Productivity: Mediating Role of Knowledge Sharing. 12. 121-135.

Varpio, L., & Ajjawi, R., & Monrouxe, L.V., & O'Brien, B.C., Rees, C.E. (2017): Shedding the cobra effect: problematising thematic emergence, triangulation, saturation and member checking. Med Educ. 51(1):40–50.

Vasilescu, B., & Serebrenik, A., & Brand, M. (2013): The Babel of Software Development: Linguistic Diversity in Open Source. 8238. 391-404. 10.1007/978-3-319-03260-3_34.

Wachs, J., & Nitecki, M., &Schueller, W., & Polleres, A.(2022): The Geography of Open Source Software: Evidence from GitHub, Technological Forecasting and Social Change, Volume 176,2022,

Wagner, S., & Ruhe, M. (2018): A Systematic Review of Productivity Factors in Software Development, Cornell University arXiv. DOI: 1801.0647

Wang, Z., & Xiao, G., & Zhang, Z., & Wu, S. (2021): A Novel Model for Automatic Identification of Open Source Software License Terms, 2021 IEEE 4th International Conference on Computer and Communication Engineering Technology (CCET)

Wilson, G. (1999): Is the open source community setting a bad example? IEEE Software, 16 (1),23–25.

Zachman, J. (1987): A Framework for IS Architecture, IBM Systems Journal (26:3), 1987, pp. 276-292.

Zhizhong, Z.Z., & Vidyanand, C.(2022): Impact of Competition from Open Source Software on Proprietary Software, DOI 10.1111/poms.13575

Appendices

Appendix A: Interview Guide

Demographics

1. Can you pleas tell me a little bit about yourself (nationality, age, employment status, profession)
    - If you are a student, what level of education have you attained?
    - If you are employed, what is the name of the Company you work for?
    - If you are a Software developer, for how long have you being working in the field?

Free and Open Source Software

1. What is free and open source software?
    - How do you perceive free and open source software?
2. Have you ever contributed to any free and open source software development projects in open source communities?
    - What are your motivating criteria for contributing in free and open source software projects?
        - Were these criteria important to you? (**Enhance programming skills** , **Programming satisfaction, Knowledge sharing, Job opportunities, Personal reputation, Earning money)** How important was each of them?
    - So, which of the criteria you mentioned do you think is/are the most important? Why?
3. Have you ever modified or enhanced any free and open source software to suit your project(s) needs? If yes
    - What was the purpose (Work related, Personal, University related projects)
    - For how long have you being modifying or enhancing free and open source software to suit your needs? Do you trust free and open source software?
    - In (approximately) how many software projects did you modify or enhance free and open source software to suit you needs? Can you tell me a little bit more about one or two project(s) where this was the case?
    - Did you encounter some problems in the project(s) when you modified or enhanced free and open source software? What kinds of problems?

4. What motivating criteria influenced your decision to modify or enhance the free and open source software to suit your project(s) needs?
    - Were these criteria important to you? (**Open source codes**, **Low or no license fees, Increases productivity, Reusability**, **Higher stability, Better functionality, Reliability)** How Important was each of them?
    - So which of the criteria you mentioned do you think is/are the most important? Why?
5. Did you adhere to the free and open source software license by publicly sharing the modified or enhanced version of the software source code based on the license agreement?

Productivity

1. Is productivity important to you personally when you develop software?
    - What do you mean by productivity or how do you perceive yourself as being productive?
    - Is your productivity important to your client/s? How?

2. In one or two of the software project(s) you modified or enhanced free and open source software to suit your needs,
    - What was the expected duration of the software project(s)?
    - How long did it take you to complete the software project(s)?
    - Without modifying or enhancing the free and open source software to suit your project needs, how long would it take to complete the software project(s) if you were to develop the software from scratch?
3. How will you evaluate the quality of your modified or enhanced version of the free and open source software?
    - What do you take into consideration when marking your software projects as complete ? (project goals, hours worked or both). Why?
4. How often do you feel you reach a high level of productivity when you modify or enhance free and open source software to suit your project(s) needs?
    - What is your overall productivity in getting your software projects completed?
5. Is there anything more you think I need to know to understand what free and open source software means to you and how it impacts your productivity?