Constructing a Theoretical Framework for Open-Source Software Projects*

Information Management Master Thesis

Tilburg University School of Economics and Management

Author: T. van der Heijden
SNR: 2013169
t.vdrheijden@tilburguniversity.edu

Supervisor: P. K. Medappa
Room: K 936
p.k.medappa@tilburguniversity.edu

June 2023

Abstract

This study sets out to construct a theoretical framework that can be used to categorise Open-Source Software projects, as well as examine relevant aspects (called attributes) that influence the outcomes of said projects. The framework was constructed by adapting a model from Nolan and McFarlan (2005), which is created in the domain of information systems, into the context of Open-Source Software. Using this framework, a curated selection of Open-Source Software projects and their attributes were empirically and iteratively analysed in a case-based manner, employing a novel methodology by Berente et al. (2019). This analysis resulted in the construction of theoretical propositions, which were used to substantiate the aforementioned framework, and provide insights into the antecedents of the project's outcomes and success. Theoretical implications and directions for further research are discussed.

*I would like to express my gratitude toward professor P. K. Medappa for his continued guidance, patience and support during the writing process of this document and professor A. Saghafi and master student S. de Waard for their help in developing and shaping phase 1 of this study, along with their continued insights and suggestions.

Contents

1 Introduction **4**

2 Framework Development **6**
2.1 Innovation . . . 7
2.1.1 Incremental and Radical Innovation . . . 8
2.1.2 Innovation Operationalisation . . . 10
2.1.3 Innovation Concepts . . . 11
2.2 Sustainability . . . 12
2.2.1 Sustainability Operationalisation . . . 13
2.2.2 Sustainability Concepts . . . 15
2.3 Grid Construction . . . 17

3 Attributes of Open-Source Software **17**
3.1 Attributes Defined . . . 18
3.2 Attribute Selection . . . 18
3.3 Finalised Attributes . . . 19

4 Computationally Intensive Theory Development **21**
4.1 Firm and Project Choice . . . 23
4.1.1 Firms in Open-Source Software . . . 24
4.1.2 Projects of the Apache Software Foundation . . . 26
4.2 Data Sources and Terminology . . . 27
4.2.1 Data Sources . . . 28
4.2.2 Terminology . . . 29
4.3 Nature of Communication . . . 30
4.3.1 Iterative Data Analysis . . . 30
4.3.2 Chat-GPT . . . 31
4.3.3 Artificial Intelligence Restrictions . . . 32
4.3.4 Formality . . . 35
4.4 Work Structure . . . 36
4.4.1 Superposition and Co-work . . . 36
4.4.2 Release Versions . . . 37
4.5 Governance and Leadership . . . 39
4.5.1 Democracy, Transparency and Structure . . . 39
4.5.2 Project Documentation . . . 40
4.6 License Type . . . 43
4.7 Firm Involvement . . . 44
4.7.1 Corporate Influence . . . 44
4.7.2 Sponsorship . . . 45

5 Theoretical Implications **49**

6 Conclusion **52**

A Additional Tables **55**

References **66**

List of Figures

1 Framework Construction Process 7
2 IT Strategic Impact Grid 8
3 Parabola Developer Score 15
4 CITD Methodology Range 22
5 Theoretical Lexicon 23
6 Reviews ASF Projects 46
7 Pull Request Participants ASF Projects 47
8 Issue Participants ASF Projects 48

List of Tables

1 Framework Concepts Innovation 11
2 Framework Concepts Sustainability 16
3 Open-Source Software Strategic Grid 17
4 Open-Source Software Attributes 20
5 Open-Source Software Firms 24
6 Descriptive Statistics ASF Projects 26
7 ASF Project Choice and Stars 27
8 Data Source per Attribute 28
9 Major and Minor Releases ASF Projects 38
10 Yearly Releases ASF Projects 39
11 Open-Source Software Attributes (Updated) 49
12 Open-Source Software Propositions 51
13 Open-Source Software Strategic Grid (Updated) 52
14 Growth Rate Numerical Example 55
15 Open-Source Software Attributes (Reference) 55
16 Formality Nature of Communication 59

1 Introduction

Open-Source Software (OSS) projects (for now on also addressed as "projects") consist of source code for a particular application, such as a game, programming framework, operating system, virtual machine, etc. This code is distributed to the public, allowing free use, modification and further distribution of said code, conditional on licensing agreements (Aksulu and Wade, 2010; Ebert, 2009). OSS exists for some time now, with the concept being brought to the attention of the software development world by the likes of Richard Stallman with his GNU Project in 1983 and Linus Torvalds with his Linux Kernel in 1991. At present, adoption of OSS in the business environment is at an all-time high[1], with firms embracing the projects that the industry produces (Di Gangi et al., 2022; Linåker et al., 2016). One might ask why these firms choose to build and develop their applications on such an open basis. After all, by "going open source", one loses the ability to keep the application proprietary, in other words, in-house, and reap private benefits. The biggest reason for this is the knowledge and experience that developers[2] from outside the firm can bring (West and Gallagher, 2006). Indeed, firms increasingly use the "wisdom of the crowds" to develop application solutions by allowing outsiders to contribute (Foege et al., 2019; Kane and Ransbotham, 2016; Ho and Rai, 2017). Outsiders are willing to contribute as they have their own personal goals to accomplish: Hobbyists seek firm recognition and a sense of accomplishment (Jeppesen and Frederiksen, 2006), entrepreneurs want to use the results of the OSS projects for their own gain (Di Gangi et al., 2022) by creating social capital for the community[3] (Larsson et al., 2019), while scholars seek to reach an understanding on what OSS is and how it functions, especially in the early stages of project development (Geiger et al., 2021).

Using this wisdom of the crowds has one major advantage: It becomes easier to produce application solutions (i.e., projects) as the sheer number of eyes on the problem increases drastically (Behfar et al., 2017; Tang et al., 2021; Piva et al., 2012). Nonetheless, due to different developers contributing to all kinds of OSS projects, a pattern has emerged: OSS projects often increase in size and complexity over time (Alenezi and Almustafa, 2015; Darcy et al., 2010) and the number of OSS projects has increased drastically[4]. This creates two problems as it complicates the choice of investment, i.e., it poses the following question: What are the best projects to invest in? After all, "there is considerable information asymmetry between OSS projects and potential users[5] as project quality is unobservable to users" (Dong et al., 2019, p. 669). The first problem relates to the type of project. With the increased complexity and number of projects, it becomes harder to estimate which projects will fulfil the needs of firms, developers and users, and presents itself in the form of different questions. Will a project exist in 2 years? 5 years? How innovative will the solution be? Is this important in the first place? These questions make clear that it becomes necessary to shed light on how different projects can be categorised. For example, a firm that is in search of infrastructure or a platform to build applications on might look for something like the Apache Webserver, where they are primarily concerned with the stability of said infrastructure or project, not necessarily with its (potentially) extensive functionality. On the other hand, a developer that wants to experiment with new and emerging technologies or develop an application that quickly offers new functionalities to customers might not be that interested in the stability of his product, but more so in the

[1] See for example the reports by Red Hat (https://www.redhat.com/en/enterprise-open-source-report/2022) or Fortune Business Insights (https://www.fortunebusinessinsights.com/open-source-services-market-106469).

[2] "Developers" is used here as an umbrella term to indicate any individual that contributes to an OSS project regardless of the contents of that contribution. While there are different types of developers (that need to be distinguished from each other for this studies analysis), for now, the term "developers" suffices. Further explanation of this term and other terminology used in the first couple of sections of this study can be found in section 4.2, where it becomes of importance to make those distinctions.

[3] Here, the community can refer to the group of developers working on a specific OSS project, or more generally, developers who work in the OSS industry, irrespective of the project(s) that they work on.

[4] See for example the report by GitHub (https://octoverse.github.com/).

[5] "Users" refers here to the individuals that use the (intermediary) results of the OSS projects. They can be entities within firms (like a floor manager that decides to use an open source developed database management system) or outside firms (like hobbyists).

novelty that it offers. These differences make it clear that categorisations are necessary to give stakeholders insights into projects. They also make clear that operating in one of these ways cannot be seen as something that is “right” or “wrong”. Rather, it provides an orientation for the way of working of a project. The second problem relates to the aspects of projects. Merely looking at categorisations is not enough. After all, projects also differ in so-called “attributes”. Attributes is used as a broad term here to indicate any concepts that tell something about the different aspects that influence the success of an OSS project, like its collaboration structure, maturity, code quality, governance etc., which should be examined as well. These attributes provide finer details about the workings of a project. If, for example, a developer seeks an informal project where he can collaborate and brainstorm with peers in an informal and direct way, but is met with a rigid hierarchy that only accepts submissions of new ideas and concepts through a formal procedure, the project might not be a good fit for him. By ignoring these attributes, no meaningful conclusions about a project’s categorisation can be garnered.

In order to tackle the categorisation problem, numerous studies have tried to categorise and group OSS projects in a meaningful way based on meta-information, often in the form of categorisation schemes. Examples include the works of Iwami (2022), who used keyword networks to group projects, or Vargas-Baldrich et al. (2015), who presented an OSS tagging tool. Other works categorise and label OSS projects based on issues[6] (e.g., Cabot et al., 2015; Kapitsaki et al., 2022), different repositories[7] (e.g., Kawaguchi et al., 2004), growth rate analysis (e.g., Koch and Stix, 2008) or online profiles (e.g., T. Wang et al., 2014). Results of these studies include distinct categories (often in the form of tags) such as “algorithm”, “operating system”, “cloud computing” etc., that can be given to a group of projects who share these aspects. While this type of approach proves useful in bringing order to the large quantity of OSS projects out there, it does not provide insights into the individual project level, failing to shed light on any appropriate categorisations discussed earlier. This study will set out to close that gap by answering the following research question:

> **Research Question 1**: How can a theoretical framework to categorise Open-Source Software projects be developed?

Another group of studies tried to find an answer to the second problem by examining the aforementioned attributes of OSS projects. Examples include the work of Adewumi et al. (2019), who developed a framework for the evaluation and selection of OSS projects. They mostly focused on the quantitative aspects of the software itself, looking at portability, usability, performance, and more. Others examined OSS project attributes by looking at collaboration (e.g., Behfar et al., 2017; Peng, 2009), governance structures (e.g., Markus, 2007), reliability (e.g., Q. Wang et al., 2019; Rahmani et al., 2010), or even studied which evaluation frameworks and studies currently exist (e.g., Yılmaz and Tarhan, 2022). While this group of studies is constructive, even critical, in reaching an understanding of these attributes, one research gap remains: A complete picture that provides an overview of the attributes. To make such a complete picture, a careful examination of the present attributes found in OSS projects as well as an examination of those attributes is warranted. This study sets out to close that gap by answering the following research question:

> **Research Question 2**: Which attributes can be identified in Open-Source Software projects?

Note that in order to complete such an identification, it also needs to include a closer examination of the attributes to make sure the identification is validated and the attributes are well understood. Combined, these research questions shed light on the scientific contribution of this study. Existing categorisations schemes of OSS projects (which were

[6] Issues in this context can refer to software issues (i.e., bugs), but this does not have to be the case. It can also refer to questions about product use and how to contribute to a project, cover legal aspects of the nature of openness of the code, and many other things.

[7] Repositories are online folders hosted on dedicated platforms where the code and information about a certain project is stored.

discussed above) are functionality driven. In other words, they are functional in their categorisation of the projects but have less focus on the nature of the projects and their outcomes. Looking at a categorisation from the outcome perspective (i.e., the complete picture of attributes) allows researchers and practitioners to examine the antecedents to the success of a project and make proper judgements with regard to time and money investment.

To answer the research questions, this study is split into two phases. Phase 1 answers Research Question 1 by building a theoretical framework based on a model by Nolan and McFarlan (2005). Multiple in-depth discussions between two professors and two master thesis students were conducted in order to construct this theoretical framework, supplemented by intermediate work that spanned over two months. Phase 2 answers Research Question 2 by first examining the attributes of projects that could be identified in the current literature. Second, these attributes were further examined by conducting a case-based analysis of 8 carefully selected OSS projects using a novel methodology by Berente et al. (2019). The case-based structure and novel methodology allowed the author to empirically examine different sources of data for each attribute, analysing over 1000 comments, 200 release versions, 20 governance documents, and finally, 1500 issue and pull request discussions. The results of this analysis presented themselves in the form of theoretical propositions, which will be constructed in phase 2.

Phase 1: Theoretical Framework

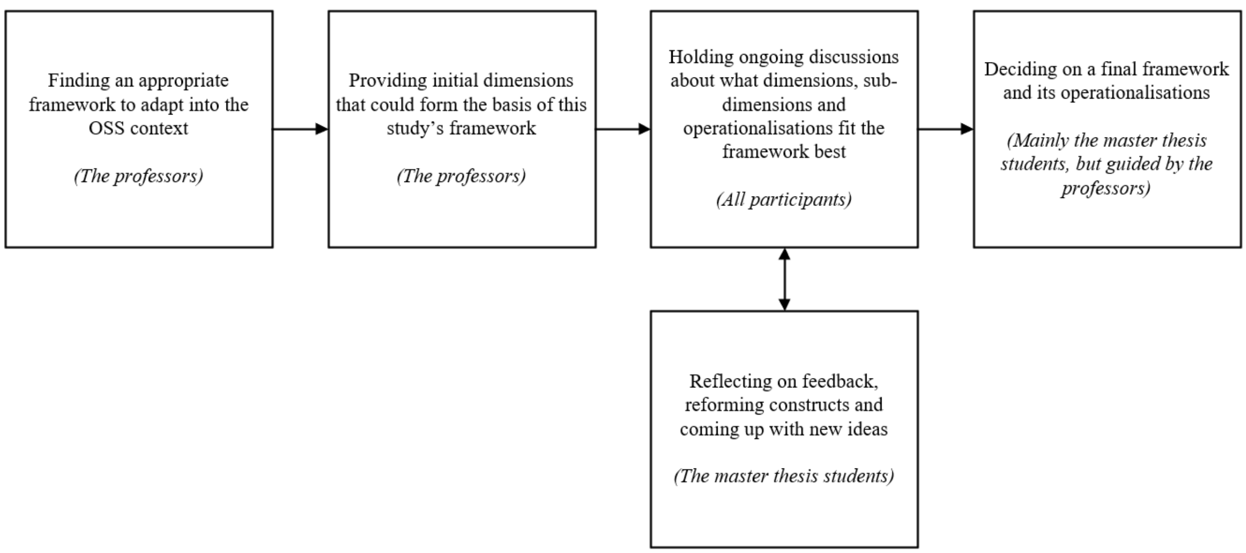
This phase outlines the construction of the theoretical framework and answers Research Question 1. As this phase solely focuses on the construction of this theoretical framework, it only has one section (titled Framework Development). The first part of this section is dedicated to specifying the process by which the theoretical framework came to be, comparable to a research methodology. The second part details the actual construction of the theoretical framework by specifying the model that was used, resulting in dimensions, sub-dimensions and operationalisations.

2 Framework Development

When constructing a theoretical framework (from now on simply addressed as “the framework”), there are no hard rules or conventions that dictate what is “right” or “wrong” to do. However, the author feels that one point should be considered. In line with the template for framework construction by Booth and Carroll (2015), the framework should be based on primary research studies of others, in addition to creating new constructs and themes. In this study, the new constructs and themes arise from adapting a model by Nolan and McFarlan (2005) into the OSS context. While this model is constructed in the domain of information systems, the author feels that it could provide a solid basis for this study’s framework for several reasons. First, the model is well-established and discussed in the information systems literature, see for example the works of Huff et al. (2006), Bart and Turel (2010) or Andriole (2009). Second, it is based on a strategic managerial perspective, which provides managers guidelines that they can follow based on their current position in the model. This aligns with the interests of a firm or developer that considers contributing to a certain OSS project, as the positioning of a project in the framework can provide insights into the current state of a project, as discussed in section 1. Thirdly, its dichotomous and grid-like structure allows for a straightforward interpretation of the different categories that arise from its construction. Finally, applying the model in its current form to the OSS context is relatively uncomplicated. This ease-of-transferability of its constructs from the information systems context to the OSS context further solidifies and grounds the framework in the existing literature.

A high-level overview (read from left to right) of the iterative process that was used to construct the framework is outlined in figure 1. Two professors and two master thesis students participated in its construction. The process started at the beginning of February, where both professors found the appropriate model to adapt into a framework that could be brought to the OSS context. Then, based on their knowledge and expertise, they provided some structure to the framework, outlining the initial dimensions. After the master thesis students joined the project, in-depth discussions were held (in the form of meetings) between all four participants on which dimensions, sub-dimensions and operationalisations (i.e., constructs) could fit the framework, which will be discussed in the sections below. Note here that one master thesis student was the author, yet the other master thesis student also participated as the finalised framework could be applied in his master thesis as well. The process of finding the right constructs took about 2.5 months, lasting until the middle of April. In the meetings, the master thesis students often proposed new ideas for the dimensions and other constructs, after which the professors provided feedback. Based on the feedback, the master thesis students would (sometimes individually, sometimes together) reform constructs and come up with new ideas that were grounded in the OSS and information systems literature. Some of this discussion (between all participants, but also between the master thesis students) occurred outside the meetings via email, resulting in an iterative, nonlinear way in which the framework came to be (illustrated in figure 1 by the bi-directional arrow between the vertically aligned steps). In the end, the framework was constructed by the master thesis students, guided by the expertise (both domain expertise and general expertise in conducting research) that the professors provided.

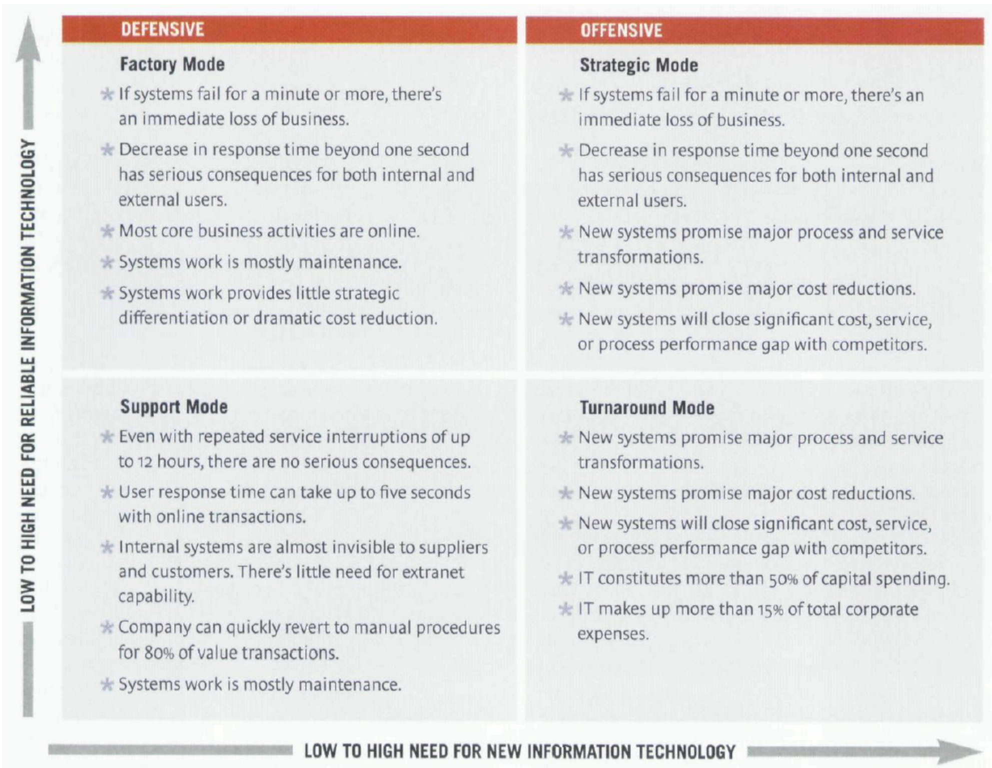
Figure 1: Framework Construction Process



2.1 Innovation

In their 2005 paper, Nolan and McFarlan explore and discuss the involvement of the board of directors in the strategic decision making process of firms. The result of their research was a 2-by-2 model, which they named the IT Strategic Impact Grid. It shows firms (and the managers of those firms) a perspective on how they are currently strategically using their Information Technology (IT) and what questions they should ask themselves based on their position in the model. This model is composed of two main dimensions: An offensive IT dimension as an x-axis and a defensive IT dimension as a y-axis. Both axes operate on a continuous scale that is operationalised using discrete categories: The x-axis ranges from low to high need for new IT, while the y-axis ranges from low to high need for reliable IT. This results in four quadrants, see their complete model displayed in figure 2[8].

Figure 2: IT Strategic Impact Grid



Of greatest interest to this study is the offensive IT dimension, which is defined in the paper as "how much the firm relies on IT for its competitive edge through systems that provide new value-added services and products or high responsiveness to customers" (p. 98). Two terms of this definition stand out, being "competitive edge" and "new value-added services". Nolan and McFarlan argue that new value-added services can be created by achieving competitive advantage through the leverage of IT. Indeed, studies have argued for this type of strategic use and confirmed it through empirical analysis (Johannessen et al., 1999; Puspitasari and Jie, 2020; Huang et al., 2005; Dibrell et al., 2008; Drnevich and Croson, 2013). Yet, they also complement or substitute the notions of new value-added services and competitive edge with "innovation". This is hardly surprising as a big part of innovation is about being new and creating value, in whatever context it may occur (e.g., Luo, 2022; Jacobides et al., 2006). The same sentiment can be garnered from the stakeholders that invest in OSS projects. After all, the creation of new, innovative application through projects is the reason for stakeholders to participate in the open-source world in the first place. Thus, in line with the x-axis of Nolan and McFarlan's offensive IT dimension, innovation will be taken as the x-axis of the framework.

2.1.1 Incremental and Radical Innovation

To keep in line with the model as much as possible, the sub-dimension of the x-axis should consist of two discrete categories. One could simply copy the given categorisations (i.e., low and high) as they work well in the context of

[8]Adapted from "Information Technology and the Board of Directors," By Nolan, R. and & McFarlan, F. W., 2005, *Harvard business review, 83*(10), p. 99.

depending on new IT. However, these categorisations do not translate well to innovation. After all, the question "what is new?", which is inherent in the innovation definition, has been asked since the 1960s (e.g., Wasson, 1960), with scholars still working toward a final, comprehensive answer (Blok, 2021). In addition, it is difficult to define what "low dependence on innovation", or more generally, "low innovation" entails. Instead of looking for ontological answers to such questions and derive complex operationalisations as a result (which falls outside of the scope of this study), another dichotomous set of categorisations is sought after. Nolan and McFarlan do provide two orientations (defensive and offensive, see the headers of the two upper quadrants in figure 2) that might be a good fit, as they further describe how IT is put to use, but can be applied to innovation as well. Defensive innovation could refer to the type of innovation that does not necessarily break the current mould but rather provides further support for existing products and services. On the contrary, offensive innovation could refer to the type of innovation that deviates from the existing way of working by introducing new products and services. However, as not to miss any potentially better categorisations, the typology of innovation was examined. Two main sources give innovation typology overviews: J. Y. Lee (2011) and Garcia and Calantone (2002)—Both discuss multiple typologies present in the literature. Categorisations that are not dichotomous (e.g., triadic categorisations, see Garcia and Calantone, 2002) or that do not make sense in the context of innovation (e.g., administrative vs. technical innovation, see J. Y. Lee, 2011) were not considered.

Three categorisation typologies stand out in this literature. The first covers adoption, and more specifically distinguishes between pre-adoption (i.e., awareness of a certain innovation) and established adoption (i.e., full commitment to the adoption decision of that innovation) (e.g., Greenhalgh et al., 2004; Wisdom et al., 2014). In the authors opinion, this is not an appropriate categorisation for innovation (at least not in the context of OSS), as adoption tells a story of practicality. If adopted, the project creates value for a user or firm (i.e., the net benefit of using the project is larger than 0, otherwise the user or firm does not adopt it) but it does not say anything about the project being new. In addition, it also cannot be translated to the design and development of a project's software, an important aspect of OSS. Similarly, the second typology (which distinguishes between collaborative and non-collaborative innovation) falls short. This typology is researched extensively in the context of OSS, see for example the studies of Behfar et al. (2017), Colombo et al. (2014), Teigland et al. (2014), and West and Gallagher (2006), who all examined the effects of collaboration structures on innovation output and firm performance. However, like the first typology, it ignores the concept of newness. In addition, results of the aforementioned studies often hint at collaborative efforts outperforming their non-collaborative counterparts, which implies that non-collaborative efforts are less suited for creating innovative outputs. This is in line with the general concept of OSS (i.e., collaboration is key), which makes a distinction between non-collaboration and collaboration innovation meaningless in this context. Finally, the third typology distinguishes between incremental and radical innovation. It is the most acknowledged and examined typology in the literature (see Garcia and Calantone, 2002). Discussing incremental and radical innovation begins by looking at the work of Souto (2015), who argues that incremental innovation is about low novelty and radical innovation is about high novelty, a notion supported by Ritala and Hurmelinna-Laukkanen (2013). Of importance here is that low novelty does not indicate a complete lack of newness—Something that is new can refer to a new way of working or of exploiting existing techniques in ways not seen before, see for example the work of Norman and Verganti (2014), who characterise it as doing better than what you already do. It can also be seen as building on existing foundations and trying to keep up with current market demands (J. Y. Lee, 2011). On the contrary, radical innovation is about producing something truly new (Souto, 2015), by doing what has not been done before (Norman and Verganti, 2014). It is about anticipating new market demands (J. Y. Lee, 2011) by outperforming the existing rate of progress (Gatignon et al., 2002). While all these sources (in addition to the model's orientations discussed before) provide very compelling perspectives on what incremental and radical innovation could entail, the definition given by Briest et al. (2020) will be taken to define what is meant by incremental and radical innovation in this study. After all, the most well-fitting and precise definition of these concepts in the context of OSS will be the best. Briest et al. (2020) define incremental innovation as quickly releasing new innovations while only providing small actual updates or changes per release. This fits perfectly in the context of OSS, as frequent updates to the code of a project usually include simple bug fixes or other small changes. In a similar vein, radical innovation is defined as slowly releasing new innovations while providing big updates or changes per release. Radical innovation is thus more concerned with fewer updates, including feature introductions or new ways in which the software operates. As such, incremental and radical innovation will be used as the categorisations and hence the sub-dimensions for innovation.

2.1.2 Innovation Operationalisation

Based on the definition of the framework's x-axis dimension (i.e., innovation) and its sub-dimensions (i.e., incremental and radical innovation), one can already make distinctions between projects by hand, i.e., using the authors judgement to define which projects are more typified by incremental innovation and which by radical innovation. However, by doing so, the author is at risk of "shoe-horning" projects into quadrants based on the limited information he has about the projects. To avoid this, the sub-dimensions were operationalised using a quantitative metric. In this way, projects are allocated to certain quadrants based on their inherent characteristics. While this might not coincide with the author's understanding of the projects, it does provide a more data-grounded view and avoids possible biases and misinterpretations. Fortunately, as the definitions of incremental and radical innovation fit the context of OSS well, operationalising these sub-dimensions is a relatively straightforward task. While there is no simple way to account for the number of updates per project (as that would require controlling variables for the projects age, size, contributor base etc.), there are two concepts that give us insight into the content of those updates: Commit frequency and the number of file changes. To explain these numbers, one must have a basic understanding of how OSS development works. When a developer sees an interesting OSS project (e.g., on GitHub[9]) that he would like to work on, he can copy the current state of that project's code to his own computer (i.e., his local machine). This code consists of different files, which are often dedicated to singular features or functions. As the developer makes changes to these files, he uses commits to track his work. A commit is a capture of the current state of the code, in other words, of the current states of the files. One can view it as a "checkpoint" or "save file". The developer could make many commits, for example where each commit is dedicated to creating a new piece of code, or he could make very few (as few as 1) commits, bundling all his work in a low number of save files (or a single one). In this process, he can alter one file, or many files depending on the work that he is doing. Once the developer is satisfied with his additions to the project on his local machine, he bundles the commits and submits them as a pull request to the platform where the project is hosted. This pull request contains the changes made to the files and the commits to make those changes. In addition, it contains the request to merge the changes into the project, effectively accepting his version of the code as the new standard. While the asynchronous development cycle on platforms like GitHub is more complicated than is displayed here (mainly due to work continuing on the platform while the developer is at work, creating multiple versions of the project), this explanation suffices for this study. Based on this development process, incremental innovations can be characterised by a low number of commits and file changes (probably only improving on very minor things or fixing bugs), while radical innovations can be characterised by several commits and file changes (to accommodate the new functionality). Both numbers (frequency of commits and number of file changes) are thus necessary to reach a conclusion on whether a project is classified as more incremental or more radical in its innovation. This results in the following operationalisation formula, which is named the "Innovation Score":

[9] GitHub is an example of a code hosting platform where geographically dispersed individuals can collaboratively work on code for applications. The platform provides the functionalities that the individuals need in order to make such a development environment feasible, like different means of communication, collaboration and task allocation.

$$Innovation\ Score = \sqrt{x * y} \quad \text{With} \quad x = Average\ number\ of\ commits \quad \text{And} \quad y = Average\ number\ of\ file\ changes$$

A higher score indicates a more radical project. The score includes a square root to normalise the results to make it more readable and comparable to the other metrics, also see section 2.2.1. Note that this metric is at the project level, as it includes the average number of commits and file changes of all pull requests. This way of measuring the type of innovation for a firm is derived from the author's own understanding of innovation within the context of OSS, as no better innovation metric could be found within the present literature—They either provide a dispersed range of aspects that deal with innovation in the context of OSS (e.g., Hwang et al., 2009; Banks et al., 2022), or actual innovation metrics (e.g., Radziszewski, 2020) that do not translate to the context of OSS. This score captures the current state of a project, not its evolution over time (excluding the temporal aspect). With this Innovation Score per project in place, a way to categorise a project as incremental or radical is sought after. Ideally, one would use some kind of benchmark—If a project's Innovation Score falls below the benchmark, then it is categorised as incremental, and if it falls above the benchmark, it is categorised as radical. Two simple yet effective options exist here: That of the median and that of the average. To illustrate the differences between the two, a fictional example is given: A firm only has 7 projects, of which the Innovation Scores are represented by the following array: [1, 1, 1.5, 2, 2, 2.5, 2.5, 15]. If one takes the median (which is 2), projects 1 through 4 would be categorised as incremental and projects 5 through 7 as radical. Instead, taking the average (which is 3.64), projects 1 through 6 would be categorised as incremental and project 7 as radical. There is no right or wrong approach here, but the author chooses to take the average as the benchmark. While the median is better at spreading out the projects among the sub-categorisations of incremental and radical, the average provides a "truer" representation of the data, including its (possible) outliers. After all, the study aims to find the differences between these categorisations (i.e., incremental and radical) of projects, so a construction that favours to highlight the contrast between the two categories is preferred.

2.1.3 Innovation Concepts

To not overwhelm the reader and provide an overview of the concepts discussed in the section on innovation and the creation of the framework, table 1 is provided. It outlines the different concepts, their definitions and operationalisations, and the literary references used in their construction.

Table 1: Framework Concepts Innovation

Concept	**Definition and Operationalisation**	**References**
Innovation	*Definition*: Creating something new that adds value, in whatever context of shape that may take place. *Operationalisation*: This dimension is operationalised through its sub-dimensions, incremental innovation and radical innovation.	Johannessen et al. (1999) Puspitasari and Jie (2020) Huang et al. (2005) Dibrell et al. (2008) Drnevich and Croson (2013) Luo (2022) Jacobides et al. (2006)

Continued on next page

Table 1: Framework Concepts Innovation — Continued

Concept	Definition and Operationalisation	References
Incremental innovation	*Definition*: Quickly releasing new innovations while only providing small actual updates or changes per release. *Operationalisation*: $Innovation\ Score = \sqrt{x * y}$ $x = Average\ number\ of\ commits$ $y = Average\ number\ of\ file\ changes$ By calculating the Innovation Scores for a number of projects, the average value of those scores can be taken as a benchmark. Projects that score lower than this average are typified as incremental.	J. Y. Lee (2011) Garcia and Calantone (2002) Souto (2015) Ritala and Hurmelinna-Laukkanen (2013) Norman and Verganti (2014) Gatignon et al. (2002) Briest et al. (2020)
Radical innovation	*Definition*: Slowly releasing new innovations while providing big updates or changes per release. *Operationalisation*: $Innovation\ Score = \sqrt{x * y}$ $x = Average\ number\ of\ commits$ $y = Average\ number\ of\ file\ changes$ By calculating the Innovation Scores for a number of projects, the average value of those scores can be taken as a benchmark. Projects that score higher than this average are typified as radical.	J. Y. Lee (2011) Garcia and Calantone (2002) Souto (2015) Ritala and Hurmelinna-Laukkanen (2013) Norman and Verganti (2014) Gatignon et al. (2002) Briest et al. (2020)

2.2 Sustainability

Section 2.1 mentions the use of a defensive IT dimension as the y-axis of Nolan and McFarlan's model. They define this as "how much the company relies on cost-effective, uninterrupted, secure, smoothly operating technology systems" (p. 98). As with the definition of the offensive IT dimension, some terms stand out, namely "uninterrupted" and "smoothly operating". These terms can be translated to the context of OSS (and software development in general) by drawing striking comparisons between uninterrupted and "stable" in addition to smoothly operating and "reliable". Relying on reliable and stable IT systems is of strategic importance to firms, as highlighted by Anwar and Masrek (2013), Drechsler and Weißschädel (2018), and Tworek et al. (2019). Stability and reliability have been of interest to researchers who want to understand the dynamics of OSS. The current literature on this topic focuses on examining issue data, which are almost always expressed in numerical terms. For example, Bouktif et al. (2014) used Bayesian Classifiers to predict code stability, while Alenezi and Khellah (2015) examined the architecture stability of systems by looking at package dependencies. Other sources, such as Rahmani et al. (2010), Singh et al. (2017) and Kumar et al. (2019), used Non-Homogeneous Poisson Process models to assess the reliability of software. These sources make it clear that studying software stability and reliability are complicated tasks that justify complete studies on their own. In addition, relying on issue data to measure the stability or reliability of a project comes with another problem: Issues on platforms like GitHub (where the code of a particular project is hosted) not only consist of bugs and code issues, but also implementation issues, user experience issues, feature requests, etc. Often, labels are present to indicate what type of issue is addressed (e.g., the label says "Type: Bug", see Facebook's React project on GitHub: https://github.com/facebook/react), but these labels are not consistent across projects (there is no universal standard or guideline) and are ill-maintained (not all issues are assigned labels). As a result, it becomes difficult to define what is meant by an "issue".

Therefore, instead of using either stability or reliability as the dimension for the y-axis, another dimension is sought after. To keep this other dimension as close to the original vision of Nolan and McFarlan as possible, the terms uninterrupted and smoothly operating were re-evaluated. These terms can be interpreted as a project running error free and without complications (which is essentially what stability and reliability entail) but one other perspective can be used as well: That of sustainability. Sustainability is usually discussed from an environmental standpoint and the actions that humans have to take in order to preserve that environment, but it can also be applied to the OSS context. After all, different OSS projects have different life-spans and fluctuations within those life-spans. The terms addressed earlier (uninterrupted and smoothly operating) apply to sustainability, as a system that is sustainable strives to perform well at any given point in time and retain that performance in the future. So (mostly) in line with the y-axis of Nolan and McFarlan's defensive IT dimension, sustainability will be taken as the y-axis of the framework.

2.2.1 Sustainability Operationalisation

Similar to the innovation dimension, the sub-dimensions of this axis also need to consist of two discrete categories. Unlike the literature on innovation, however, there are no dichotomous categorisations present in the literature for software sustainability (or OSS sustainability for that matter) that the author could find. As a result, it was up to the author himself to decide how to best approach such a categorisation. It was decided to go for a very simple yet effective categorisation of "sustainable" and "unsustainable" to keep them in line with the dichotomous categorisations found in the model of Nolan and McFarlan as much as possible. However, categorising the sub-dimensions this way does pose an important question: How can this be operationalised? In other words, how do we measure what we mean by sustainable and unsustainable? After all, similar to the innovation sub-dimensions, the author wants to avoid making any premature assumptions about the sustainability of a project. Finding an answer to this question involves examining a project's developer base. For an OSS project to exist, it needs developers to contribute to the project. Without developers, there is no development of the code and no product to be build or further refined. Thus, the sustainability of the project is influenced by the rate with which a project attracts new developers who want to contribute and avoids a decrease in the total developer base (as this is harmful to the project), which is confirmed by Chengalur-Smith et al. (2010), Eckert and Mueller (2017), Schweik (2013), Liao et al. (2018), and Ye et al. (2008). As such, projects that have a total developer base that grows more relative to his peers are considered more sustainable. This results in the following operationalisation formula, which is named the "Growth Rate":

$$Growth\ Rate = \frac{(\frac{x_2}{x_1} + \frac{x_3}{x_2} + \frac{x_4}{x_3} + ... + \frac{x_i}{x_j})}{j}$$

With x being the number of unique developers who committed to the project in that specific year, excluding those who did not contribute anything. The subscripts represent the years. So, x_1 represents the number of unique developers in year 1, x_2 the number of unique developers in year 2, and so on. The total of the numerator is divided by j, which indicates the number of years the project has been growing. This is done to average the results. Table 14 provides a fictional numerical example of calculating this metric which can be found in Appendix A: Additional Tables.

However, the attraction and retention of new developers is not the only factor that determines the sustainability of a project. Activity within a project is also of importance when examining the sustainability of a project, as confirmed by Stewart, Darcy, et al. (2006). More specifically, the type of activity is important, which can be tied to the type of developer contributing to the project. Different sources distinguish between two types of developers who execute different tasks. The first type consists of developers who are very involved in the development process and contribute the bulk of the code, often in the form of new or enhanced functionality. In the literature, they are referred to as core contributors (Chengalur-Smith et al., 2010; Long and Yuan, 2005; Eckert and Mueller, 2017; Ye et al., 2008; Crowston and Shamshurin, 2017). In a later section (specifically, section 4.2), the term "contributor" will be further explained and used to show how "contributors" and "developers" are not the same. For now, the term "developer" will be used to address a "contributor" and can be understood as an individual that contributes code to the project. As such, the term "core contributor" in this study will be understood as "core developer". In the same sources mentioned before, the second type of developers is highlighted, called peripheral contributors. For the same reason as before, for now, the term "developer" will be used as a replacement for "contributor", and as such, "peripheral contributors" will be addressed as "peripheral developers". These peripheral developers are usually less involved, contribute a marginally smaller amount of code, and do not work on the functionality of the project. Instead, they concern themselves with bug fixing, general problem solving, and testing of the software. While the previously mentioned sources argue that both types of developers are necessary to reach and retain a sustainable project, little to no attention has been given to the exact or optimal ratio of core developers versus peripheral developers. On the one hand, a low number of core developers (with several peripheral developers) results in the projects not actually being developed, as developing and integrating new features rests on the shoulders of the core developers. This is especially the case for smaller projects. On the other hand, many core developers (with a low number of peripheral developers to match) results in an ill-maintained project where bugs and problems are not addressed. As such, the author opts for a balance between the two, resulting in the following operationalisation formula, named the "Developer Score":

$$Developer\ Score = x^{0.5} * (1 - x)$$

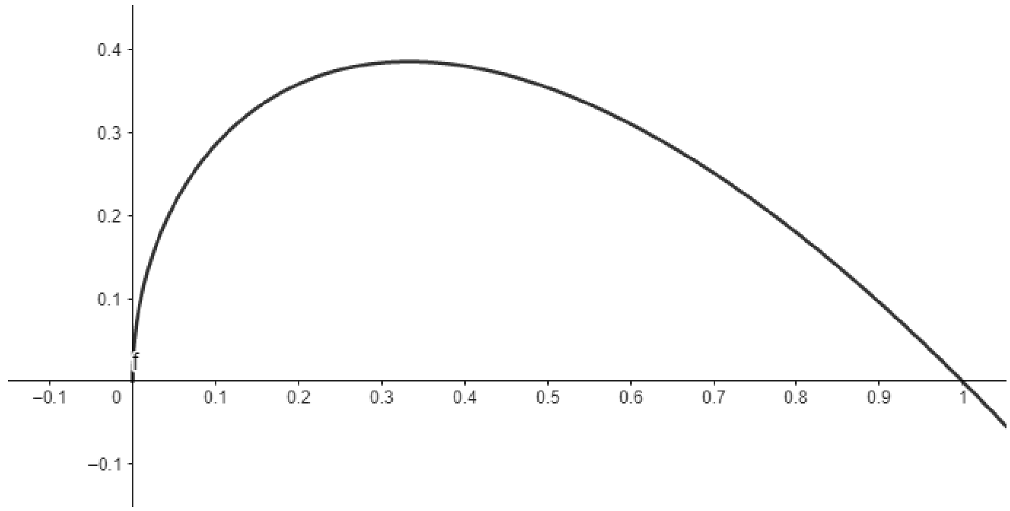
Where $(1 - x)$ represents the Peripheral Developer Ratio and x the Core Developer Ratio. The Core Developer Ratio is defined as follows:

$$Core\ Developer\ Ratio = \left(\frac{number\ of\ core\ developers}{number\ of\ developers}\right)$$

From this it follows that the number of developers = number of core developers + number of peripheral developers. Core developers are defined here as the least number of developers that wrote 80% of the total commits, which is in line with the definition given by Ye et al. (2008). Defining the Developer Score like this results in the formula being equal to a skewed parabola that rewards projects that have a x (so, a Core Developer Ratio) between 0.2 and 0.5 most, see figure 3. The choice to reward projects that have between 20% and 50% of their total developers being core developers the most is based on the logic that a project should not have too little core developers, nor have too many.

Similar to the operationalisation for the sub-dimensions from section 2.1.2, these operationalisations (Growth Rate and Developer Score) are derived from the author's own understanding of sustainability within the context of OSS.

Figure 3: Parabola Developer Score



Unlike the section that discussed the innovation operationalisation, both scores have an inherent temporal aspect to them (both measure the change of something over time) but do so for the entirety of the project. In this way, they are consistent with each other. In addition, the base idea of these operationalisations was derived from the Vitality Score of a project given by De Noni et al. (2011):

$$Vitality\ Score = \sqrt{\frac{releases * age}{lastrelease}}$$

This score could not be taken on itself as the operationalisation of sustainability, as it does not include the concepts of core and peripheral developers. In addition, while the relationship between the number of releases and the last release initiated the idea of using developer ratio's, it cannot be controlled by the age of the project alone, as the size of the project is also indicative of the number and frequency of releases, which was something that could not be controlled for in this study.

In contrast to the innovation operationalisation (which consisted of one metric, the Innovation Score), the operationalisations for this axis are twofold (the Growth Rate and the Developer Score). Therefore, the approach that was taken for determining if projects were incremental or radical (by looking at the average value across all the project's Innovation Scores) cannot be taken here. Note that combining the averages of the two metrics for this axis into some kind of overarching average, for example by adding the average value of several projects for Growth Rate to the average value of several projects for the Developer Score and dividing by two, could skew the value in favour of one of the two metrics based on their inherent scales. Instead, a more separated way of determining which projects are sustainable or unsustainable is taken: If a project's Growth Rate is above the average for all projects and a project's Developer Score is above the average for all projects, then a project is considered sustainable, otherwise it is not.

2.2.2 Sustainability Concepts

Similar to the innovation section, an overview of the concepts discussed for the construction of the framework in this section is provided in the form of table 2. It (once again) contains the concepts, their definitions and operationalisations, and the literary references used in its construction.

Table 2: Framework Concepts Sustainability

Concept	Definition and Operationalisation	References
Sustainability	*Definition*: Creating something that performs well, and holds that performance in the future. *Operationalisation*: This dimension is operationalised through its sub-dimensions, sustainable and unsustainable.	Bouktif et al. (2014) Alenezi and Khellah (2015) Rahmani et al. (2010) Singh et al. (2017) Kumar et al. (2019) Anwar and Masrek (2013) Drechsler and Weißschädel (2018) Tworek et al. (2019)
Unsustainable	*Definition*: An unsustainable project (as defined by the operationalisation). *Operationalisation*: $Growth\ Rate = \frac{(\frac{x_2}{x_1}+\frac{x_3}{x_2}+\frac{x_4}{x_3}+...+\frac{x_i}{x_j})}{j}$ $Developer\ Score = x^{0.5} * (1-x)$ $Core\ Developer\ Ratio = \left(\frac{number\ of\ core\ developers}{number\ of\ developers}\right)$ If either a projects Growth Rate is below the average for all projects or a projects Developer Score is below the average for all projects, or both are below the average for all projects, then a project is unsustainable.	Chengalur-Smith et al. (2010) Eckert and Mueller (2017) Schweik (2013) Liao et al. (2018) Long and Yuan (2005) Ye et al. (2008) Stewart, Darcy, et al. (2006) De Noni et al. (2011)
Sustainable	*Definition*: A sustainable project (as defined by the operationalisation). *Operationalisation*: $Growth\ Rate = \frac{(\frac{x_2}{x_1}+\frac{x_3}{x_2}+\frac{x_4}{x_3}+...+\frac{x_i}{x_j})}{j}$ $Developer\ Score = x^{0.5} * (1-x)$ $Core\ Developer\ Ratio = \left(\frac{number\ of\ core\ developers}{number\ of\ developers}\right)$ If a projects Growth Rate is above the average for all projects and a projects Developer Score is above the average for all projects, then a project is sustainable.	Chengalur-Smith et al. (2010) Eckert and Mueller (2017) Schweik (2013) Liao et al. (2018) Long and Yuan (2005) Ye et al. (2008) Stewart, Darcy, et al. (2006) De Noni et al. (2011)

2.3 Grid Construction

The product of the framework is displayed in table 3, which displays the full framework in its grid-like structure. The framework is constructed in the context of OSS, so it is named the Open-Source Software Strategic Grid, keeping it in line with the model of Nolan and McFarlan (2005). As this study progresses and analyses the attributes mentioned in section 1, the Open-Source Software Strategic Grid will be referenced (by addressing table 3) to give context to those findings. Once the research process is complete (i.e., phase 2 is finished), the Open-Source Software Strategic Grid will be complete, as it not only visually represents the framework but also the end-product of this study (which can be found in section 5). For now, it can answer Research Question 1: How can a theoretical framework to categorise Open-Source Software projects be developed?

Table 3: Open-Source Software Strategic Grid

		Innovation	
		Incremental	*Radical*
Sustainability	*Sustainable*		
	Unsustainable		

Phase 2: Attribute Analysis

The second phase of this study tackles the identification and analysis of the attributes and answers Research Question 2. Contrary to phase 1, this phase consists of multiple larger sections as there are more distinct and complex steps involved in the process. First, section 3 discusses the attributes that were found in the literature and explains how the author decided on the scope of this phase of the study, determining which attributes would be examined in-depth. Second, section 4 outlines the methodology, analysis and results, which requires some explaining about its structure. The first part describes the methodology that was used in this phase of the study. The second part outlines section 4.1, which justifies the choice for the selection of the examined firm and projects, while section 4.2 specifies the data sources that were used and explains some relevant terminology. The third (and final part) covers sections 4.3 to 4.7, where each attribute that was selected for in-depth analysis in section 3 is analysed, and subsequent results of that analysis are shown. It was chosen to put all components of methodology, analysis and results within these sections to accommodate the iterative approach that was taken in the methodology, see the first part of this section. Based on the results from this part, theoretical propositions are formed. The final section in this phase (section 5) functions as an overview of this phase, and combines the Open-Source Strategic Grid from phase 1 with the results of this phase (i.e., the propositions) to produce the end-product of this study and discuss its theoretical implications.

3 Attributes of Open-Source Software

With the Open-Source Software Strategic Grid laid out in section 2.3, it is possible to shift the focus to the OSS project's attributes. Once again, the attributes of OSS are examined to garner a complete picture of the environments in which projects operate and to gain an understanding of the antecedents of success. To find the attributes of OSS projects, the current literature on the topic is to be examined, which is the aim of this section.

3.1 Attributes Defined

Investigating the attributes of OSS is a challenging task for two main reasons. First, one must have an understanding of what is meant by attributes. In this study, attributes indicate any metric or concept that tells something about the different aspects that influence the success of an OSS project. In other words, one can ask themselves the following question to get a conceptual idea of what an attribute entails: What drives the success of OSS projects? It was chosen to define attributes in this way in order to not miss out on any potentially interesting concepts, scope out the definition to a manageable degree and keep it in line with the current literature on the topic. For example, studies such as Crowston et al. (2012) and S.-Y. T. Lee et al. (2009) operate in this way. If the definition was defined any broader (e.g., without the “success” component present in the current definition), one could look at the entire landscape of OSS development, which is too big a task for this study. Note here that this definition does not distinguish between different types of success, as other studies have (see for example Midha and Palvia, 2012). Second, scoping out the attributes themselves is a challenge, as a too tightly defined attribute might miss out on interesting constructs and information (see sections 4.3 to 4.7), while a too broadly defined attribute leaves the author with an infinite amount of work and ill-defined propositions.

3.2 Attribute Selection

To overcome these challenges, the paper by Crowston et al. (2012) was taken as a baseline on how the identification and substantiation of the attributes could be done. Their organizing framework is based on the Input-Mediator-Output-Input model of Ilgen et al. (2005). It shows how different attributes they found in their literature review can be placed in that model. While it is beyond the scope of this study to attempt a similar approach (which requires a systematic literature review[10]), one other element of the study can be considered: The Level-Of-Analysis (LOA). To bring structure to the collection of papers examined in their study, the following perspective on the LOA was taken: “We distinguished among studies at the artifact (which captures papers whose focus is on source code of FLOSS[11], technologies that support FLOSS development such as SourceForge[12], and programming or algorithms), individual, group, organization, and societal levels” (p. 7:6). This LOA chain was adopted into this study, resulting in table 15. It displays the LOA along with the attributes found in the literature per level and a short and rough description of the attributes. While this table is by no means exhaustive (or polished), it serves as a reference for constructing later tables. It can be found in Appendix A.

While the table and its LAO provide a compelling overview of the attributes found, not all of them can and will be examined more deeply in this study. For once, attributes that are only validated through one source are not deemed generalisable, as no other studies have confirmed their existence (yet). The discovery of the attribute could be influenced by the studies design, where another outcome could have been found if the approach differed. Thus, these attributes are not considered. Second, one must view the table through a lens of scope and practicality. To identify the most relevant factors for the success of a project, it was chosen to focus only on the development process of that success, not on the actual content of its success (its code) or its subsequent success outcomes (its adoption and use). While the author feels that these are important concepts, assessing software quality on its own is challenging enough, also see the discussion at the beginning of section 2.2 that talked about software reliability and stability. Similarly, investigating the adoption and user dynamics of a project requires a broad scope and several data sources, for example looking at download numbers, but also finding measures for user satisfaction, interest and net benefits, see table 15,

[10] It was chosen to not conduct a systematic literature review in this study due to its exploratory nature.

[11] While there are conceptual differences between FLOSS, or Free Libre Open-Source Software, and OSS (mainly through FLOSS’es interpretation of “freedom”), making such a distinction falls outside the scope of this study.

[12] SourceForge is a platform similar to GitHub.

which is not feasible for this study. In addition, the attributes present in the societal levels are often taken as the outcome measurements of success in other studies (e.g., Midha and Palvia, 2012; Sen et al., 2012), which would conflict with the aim of this study, as it tries to find the underlying reasons for those outcomes, not the outcomes themselves. As a result, the artifact LOA and the societal LOA are omitted from the table. Omitting these LOAs has the added benefit that the remaining LOAs (individual, group and organisation) translate wonderfully to the context of OSS code hosting platforms like GitHub, where rich information about certain projects can be found, drawing parallels between the individual level and developers, the group level and projects, and the organisation level and the firms that support the project. Finally, some attributes were omitted by hand due to various reasons. Developer size, growth, and ratio of core vs. peripheral developers are used in the construction of section 2's framework, being examined regardless of attribute identification and hence are removed. Developer motivation and interest to participate in a certain project would need to be examined through qualitative, text-based data such as interviews, which are hard to collect for the author. A developers ability to fix bugs could be quantified in some way, for example by looking at the time between a pull request containing an issue or bug opening and closing[13], but this metric is affected by some other related factors. These other factors include the number of developers, where a higher number of developers provides more opportunities for someone or a group of people to pick up the issue and start working on it, the level of expertise of the developers, the urgency of the issue or bug and more. This is why it was chosen to remove the developers ability to fix bugs. Finally, successful project initiation was removed for a couple of reasons. First, it receives little support from the literature, see table 15 in Appendix A. Second, as stated before, the aim of this study is not to define success but to look at its underlying factors. Finally, after some further analysis of the current literature, it became apparent to the author that defining what "success" or a "successful project initiation" entails is extremely difficult and justifies a separate study on its own, see for example the works of Conlon (2007), Thacker and Knutson (2015) or Gamalielsson and Lundell (2014).

3.3 Finalised Attributes

After the narrower selection of attributes was made, they were re-evaluated to provide a more defined scope and definition for each attribute. Based on some discussion among this study's research participants (the author and the professors), some changes were made. First, the attribute of developer coordination and collaboration could be split into two—One part focusing on the nature of communication and one part on the work structure. The nature of communication focuses on the communication style employed within the project. This can range from highly informal chatter and remarks on pull requests and issue discussions to rigid and formal means of communication. Parallels between such a range of communication styles and the governance and leadership of a project (and even a firm) can be made, but one is on the individual level, one is on the group level, and one is on the organisational level. Work structure indicates the style of working within a project. It ranges from incremental and individual work (called superposition, e.g., see Medappa and Srivastava, 2019 or Howison and Crowston, 2014, not to be confused with incremental innovation) to collaborative and co-work, also see section 4.4. The governance and leadership attribute was re-defined to include three components from the studies of O'Mahony and Bechky (2008) and O'Mahony and Ferraro (2007): Democracy, transparency, and structure. In short, democracy covers how (non) autocratic a project is run, transparency details how much documentation is available on governance and leadership, and structure defines how extensive the agreements of governance and leadership are within the aforementioned documentation. For a more extensive discussion, see section 4.5. The definition of license type remained largely unaltered, but was rephrased to be more concise and to-the-point. Finally, the definition of firm involvement was refined as well. Note here that the

[13] Note here that a pull request can cover the fixing or addressing of an issue, but this does not have to be the case.

involvement of a firm in the projects that will be chosen (see later section 4.1) can come from two sources. The first concerns the firm to which the projects belong. Thus, this concerns a "top-down influence" by the management of the firm and the control it wants to (not) exert on the projects. The second concerns influence from firms other than the firm that owns the projects, in the form of sponsorships. Sponsorships can be given in two ways. First, the outside firm can sponsor the firm that owns the projects directly, and the firm that owns the projects determines how this monetary contribution will be spent, and how much will end up at the projects. Second, the outside firm can sponsor (i.e., pay) its internal developers to work on specific projects (which is possible as the projects are Open Source), in the hope of gaining influence in where a specific project is headed, or speed up the development of the project in general so that the firm can profit from its functionality sooner. These different forms of sponsorship are critical drivers in the success of the project, either directly or indirectly (Y. Wang et al., 2022; West and O'mahony, 2008; Santos Jr et al., 2011; Stewart et al., 2005). As a final note, keep in mind that the control that the firm exerts on his projects is on the organisation LOA here, as opposed to the group LOA of governance and leadership.

As a result, the re-evaluated and finalised attributes are displayed in table 4, as an update to table 15 (which was the product of section 3.2). Note here that the naming of the individual and group LOAs was changed from its initial construction in table 15 and the Crowston et al. (2012) paper to better reflect the OSS context. Thus, the individual LOA was renamed to the developer LOA, the group LOA to the project LOA, and the organisation LOA to the firm LOA.

Table 4: Open-Source Software Attributes

Level-Of-Analysis	**Attribute and Definition**	**References**
Developer	Nature of communication – What is the level of formality employed in the project's communication.	Crowston et al. (2012) Medappa and Srivastava (2019) Peng et al. (2013) Peng (2009) Casalo et al. (2009) Wu et al. (2016) Wu et al. (2023) Bellantuono et al. (2012) Howison and Crowston (2014)
	Work structure – What is the style of working within a project with regard to superposition and co-work.	Crowston et al. (2012) Medappa and Srivastava (2019) Peng et al. (2013) Peng (2009) Casalo et al. (2009) Wu et al. (2016) Wu et al. (2023) Bellantuono et al. (2012) Howison and Crowston (2014)

Continued on next page

Table 4: Open-Source Software Attributes — Continued

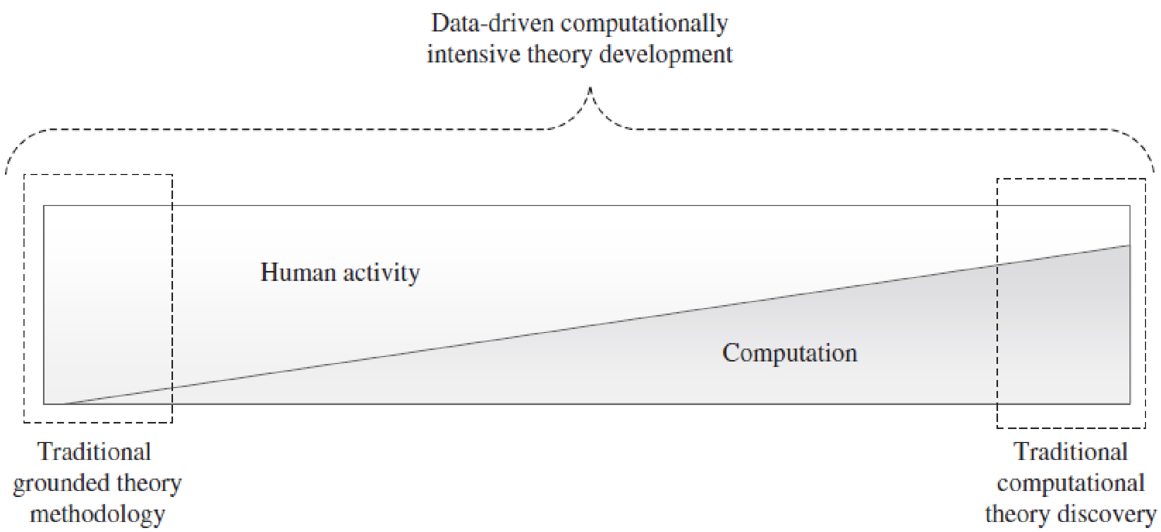
Level-Of-Analysis	Attribute and Definition	References
Project	Governance and leadership – What is the governance and leadership structure of a project with regard to democracy, transparency and structure.	Midha and Palvia (2012) Crowston et al. (2012) Sagers (2004) Neufeld and Gu (2019) Aberdour (2007)
	License type – Which license is tied to a project and how does the choice of license influence the project's outcomes.	Midha and Palvia (2012) Crowston et al. (2012) Gezici et al. (2019) Amrollahi et al. (2014) Subramaniam et al. (2009) J. A. Colazo et al. (2005) Showole et al. (2011)
Firm	Firm involvement – What is the level of control and sponsorship imposed on projects by the firm that owns the projects and by external firms.	Crowston et al. (2012) Medappa and Srivastava (2019) Deodhar et al. (2010) Lamastra (2009)

4 Computationally Intensive Theory Development

To find an answer to the second research question in section 1, a case-based analysis of OSS projects was conducted. Using a case-based approach allows an in-depth examination of the attributes found in section 3.3 within a curated set of OSS projects. This reasoning is perfectly captured by Lindman et al. (2010): "Case studies are appropriate in creating rich descriptions in cases where the context and phenomenon are difficult to separate" (p. 240). "Context" is the most important concept here and is in line with the critical realist stance of Walsh (2014). While it is beyond the scope of this study to dive into its workings, it can relatively simply be understood as follows: "A generates B in the context of C", as opposed to "A generates B". This translates beautifully to this study, as the attributes and accompanying propositions are OSS-context dependent. To accommodate such an approach, a fitting methodology was sought after. As a starting point, the base execution of the Computationally Intensive Theory Development (CITD) method of Berente et al. (2019) was examined. This novel method combines the paradigms of the traditional manual Grounded Theory (GT) and the (almost) fully traditional automated Computational Theory Discovery (CTD) methods to analyse trace data. Trace data in the context of their study refer "to the digital records of activity and events that involve information technologies" (p. 50). This is in line with studying OSS projects through second-hand data, as discussions on pull requests, issues and other types of information on and outside platforms such as GitHub can be typified as trace data. By using components of both methods (GT and CTD), CITD derives theory in an inductive and iterative manner, whereby neither the use of only qualitative nor the use of only quantitative methods suffice, akin to using a mixed-method approach, examples of which can be found in the studies of Walsh (2014), Miranda et al. (2015), Lindberg et al. (2016), and Vaast et al. (2017). Note that this type of approach should not be confused with what is called a multi-method approach, where one method (e.g., qualitative) is used to derive theory and one method (e.g., quantitative) is used to empirically test that theory. Instead, both methods are necessary in the theory formation process. This is in line with this study's design, as all components of the methodology are used to derive theory in the form of propositions, not empirically validate those propositions. At first, the base execution of the CITD method seemed like a good fit for this study. However, after careful examination of the method, the author concluded that the basic structure of the method was not a good fit, as the use of automated quantitative techniques inherent in CTD clashes with one of the studies main objectives and research questions. Recall that the second objective of this study was to not only to identify certain attributes in OSS projects, but also to examine them in-depth. Note here that it is not the use of quantitative data and techniques that forms a problem (as empirical analysis consists of both qualitative and quantitative analysis), but the focus on automated techniques, which are hard to execute in a study that focuses on in-depth analysis of attributes.

Fortunately, as Berente et al. highlight in their paper, the CITD method is not a fixed and rigid approach, nor are there standards or rules by which the method must comply. Instead, the method operates on a continuous scale ranging from full traditional GT to full traditional CTD. This is illustrated in their paper by figure 4.

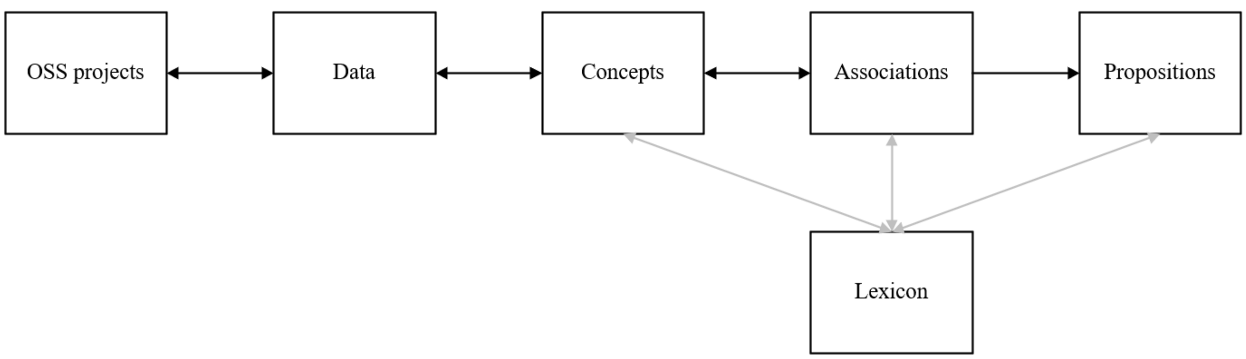
Figure 4: CITD Methodology Range



As such, this study will adapt the CITD method with its focus relatively close to the full traditional GT methodology, using relatively few CTD elements. With the GT-oriented CITD being chosen as the methodology of this study, one is able to shift the focus to the other parts of this section: The choice for a firm and projects and the attributes. Note that section 4.2, which discusses the data sources and terminology and exists in-between the sections on firm and project choice and the attributes, is not relevant for this discussion. Discussing these parts here first (before diving into their respective sections) is of importance as they illustrate how the use of both quantitative and qualitative data as well as the influence of GT and CTD is established. While the choice for a particular firm is made by examining the current landscape of OSS (also see section 4.1), the choice for the selection of projects is build on automated quantitatively collected data, using the operationalisations of sections 2.1.2 and 2.2.1, adhering to the CTD element of the methodology. On the other hand, the analysis of the attributes is done by iteratively analysing both quantitative and qualitative data, focusing more on the GT element of the methodology. Note here that while the classic interpretation of GT would form constructs and relationships among those constructs based on the actual content of the data (like in an interview, see Walsh, 2014 for an example), here, the author is not always interested in the actual content of the discussion. For example, when examining the nature of communication found in table 4, issue discussions can be analysed (see section 4.3). In this analysis, the author is interested in the formality of the discussion, not the actual discussion itself. Hence, a more loosely-formed interpretation of GT will be handled, whereby the same basic premises apply, but will be changed and formed to fit the context of the data that is examined. After all, the end goal of GT is to construct theory (Walsh, 2014) with any type of data (Walsh, 2015; Lingard et al., 2008). Some final notes on the use of GT include the focus on substantive theory formation, as opposed to formal theory formation (it only functions within a specific area of inquiry), yet strives toward rupture theorising (as opposed to incremental theorising) within that area of inquiry[14].

As a final thought, while there is no golden standard for conducting the iterative part of CITD (which will be the analysis of the projects in this study), there are some steps from GT and CTD that are altered and combined to fit the CITD way of working. These steps can be found in table 1 of the Berente et al. paper (p. 56), being Sampling and Data Collection, Synchronic Analysis, Lexical Framing and Diachronic Analysis. While this study will not strictly adhere to these steps (as some do not fit the research conducted for the individual attributes), the Lexical Framing step will be taken and adapted to fit the contents of this study. Figure 1 in the Berente et al. paper (p. 52) outlines the role of the lexicon in the generation of theory, and as such will be adopted into the context of this study, the result thereof being presented in figure 5.

Figure 5: Theoretical Lexicon



Instead of analysing the world, only a selection of projects is taken. The data presents itself in the form of actual data extracted from platforms like GitHub (but can be gathered from other sources as well, see section 4.2) and attributes from the literature. Concepts and associations are directly taken from the existing figure, both outlining how the author will form concepts of and associations between constructs when deriving theory, i.e., the propositions. As per the figure of Berente et al., “black arrows describe the iterative process of building theory, with the general direction going from empirical to theoretical; gray arrows describe referencing to and from the lexicon in this process.” (p. 52). Note here that while the current studies figure is rotated for visual clarity (here read from left to right), the direction and orientation of the arrows have not changed. Thus, the general process of deriving theory still starts with the world (in this studies case, the OSS projects) and ends with theory formation (in this studies case, the propositions) in an inductive and iterative manner.

4.1 Firm and Project Choice

To conduct a case-based analysis of the attributes, two projects per quadrant of table 3 (found in section 2.3) were examined. The choice for 2 projects per quadrant was led by practicality and research design—Only examining 1

[14]For a discussion on incremental and rupture theorising, see Walsh (2015).

project per quadrant would not allow the author to make comparisons between projects in the same quadrant, while examining 3 or more projects per quadrant would not be possible due to time constraints and slowly shift the research focus of this part of the study away from a case-based in-depth analysis. To make a selection, a careful examination of different firms involved in OSS development was conducted.

4.1.1 Firms in Open-Source Software

The choices for certain projects were conditional on the involvement of a firm[15], and hence, the choice for a certain firm needs to be explained first. There are three reasons or general thoughts for why it was chosen to only analyse projects from a firm. First, by only looking at projects that have a firm involved, it is possible to see the different levels of organisational influence exerted on different projects and their attributes. Second, a firm must be large enough. A large firm has a portfolio of projects that differ in size and popularity, allowing for a non-homogeneous and diverse selection of projects, avoiding possible biases and non-generalisability. For example, if only extremely popular projects were selected, the formed propositions would only be able to tell something about such projects. Finally, the presence of a firm in particular ensures a certain degree of reputability and credibility, in addition to providing ample data-points in the quantitative construction of a dataset of projects. Only firms that are large enough with regard to their project portfolio are considered as contenders, which is defined here as those that host more than 250 repositories (i.e., projects) on GitHub. This arbitrary number was chosen after several firms were examined. Although even firms that have a (relatively) low number of projects (say 100) will always fill the quadrants (because an approach that utilises averages that was taken), a firm with a larger number of projects will present a more balanced average and account for possible flukes and turbulences that might be present in smaller firms. The choice to include only firms (and by extension their projects) that host their repositories on GitHub is made for the simple reason that the platform provides a breadth of information on the nature of communication, work structures, firm involvement, etc. (see for example the issues and pull request pages of the Airflow project[16]). While GitHub will not be the only source of data employed in this study (see also table 8 in section 4.2.1), it often provides a good starting point. The resulting firms can be found in table 5.

Table 5: Open-Source Software Firms

Firm	# of Repositories	Remarks
Apache Software Foundation (ASF)	2465	-
Google	2508	-
Microsoft	5578	-
Amazon	134	-
Cloud Native Computing Foundation (CNCF)	134	-

Continued on next page

[15]Note here that this does not have to be a for-profit firm.
[16]https://github.com/apache/airflow/issues;
https://github.com/apache/airflow/pulls

Table 5: Open-Source Software Firms — Continued

Firm	# of Repositories	Remarks
International Business Machines (IBM)	2863	many of the larger and more popular projects (those that received a lot of stars, which indicates that a user or developer saved a project for later use or reference) show a lot of inactivity (based on the contributions made) which could be an indicator of “dead” or underutilised projects. This could skew the propositions towards those types of projects, which should be avoided.
Intel Corporation	1040	Almost all projects seem to be focused on applications and solutions that are functional to the Intel Corporation themselves, which diminishes the range of potentially different kinds of projects that can be analysed.
Meta	126	-
Oracle	267	There is a heavy focus on the GraalVM repository and derivatives of that repository, which places the focus on one solution or application. In addition, about half of the projects that received a lot of stars are inactive, similar to those of IBM.
Red Hat	84	-
SAP	261	There is a heavy focus on the OpenUI5 repository and derivatives of that repository, which places the focus on one solution or application. In addition, about half of the projects that received a lot of stars are inactive (similar to IBM and Oracle), and it is mainly focused on repositories that are functional to SAP itself (similar to the Intel Corporation).
Ubuntu	121	-

After the exclusion of firms that do not have enough repositories, 7 firms remain: The Apache Software Foundation (ASF), Google, Microsoft, International Business Machines (IBM), Intel Corporation, Oracle, and SAP. From those 7, 4 were excluded based on further examination of their project portfolio, see the remarks column in table 5. Reasons for exclusion were many inactive projects (IBM, Oracle, SAP), a heavy focus on a specific project and derivatives (Oracle, SAP), or a focus on projects that were functional only to the firms themselves (Intel Corporation, SAP), as all of them would favour a certain aspect of the firm or a specific project, which would lessen the generalisability of the analysis. This leaves 3 firms: The ASF, Google, and Microsoft. Of these three, the ASF has the most deeply rooted ties to the OSS landscape (as it is fully OSS-based) and its history[17], and hence was chosen as the firm to analyse in this study.

[17]https://www.apache.org/foundation/

4.1.2 Projects of the Apache Software Foundation

To find a selection of projects to analyse from the ASF, the operationalisations of sections 2.1.2 and 2.2.1 were employed. Due to complications during the data collection process, it was not possible to collect the necessary data (the Innovation Scores, Growth Rates and Developer Scores) for all the 2.5K repositories that the ASF manages. Fortunately, this data could still be collected for up to 1K projects, which is the workaround used in this study. Note here that these are not the top 1K most popular projects, nor are they selected based on some other specific metric or requirement. Instead, 1K random projects from the ASF were taken. After the data for these projects were gathered, several projects were omitted from the dataset due to their scores being 0 (either for the Innovation Scores, Growth Rates or Developer Scores), which resulted in 863 projects remaining. Note here that the 0's for these data do not necessarily indicate that the metrics could not be calculated for the projects, but they do indicate that a project is stagnant or only has 1 developer, which makes investigating work structures etc. difficult. Hence the omittance. While the remaining projects could all be a good fit for this study, the high number of them (over 800) does not alleviate the difficulty in selecting 2 projects per quadrant. As such, stricter filters were applied to the 863 remaining projects to further narrow down the selection. First, projects that had a Growth Rate between 0 and 1 were removed, as a number between 0 and 1 for this metric indicates that a project's developer base had shrunk. As this study is interested in finding the attributes of success for an OSS project, factors that inhibit this success should be removed. A shrinking developer base falls under this category, as confirmed by Midha and Palvia (2012) and Istiyanto et al. (2009). Second, projects that were "derivatives" from other projects were removed as well. Derivatives is used here as a term to describe projects that are built on the core premise of a "parent" project, either providing enhanced functionality or performance (usually for a specific adoption of the parent project) or supporting the parent project. An example of this is the Accumulo project, which is a parent project that allows users to store large sets of key-value data in a cluster. One of the derivatives of this project is the Accumulo website, which provides information on what Accumulo is, a link to the GitHub source code, documentation, release information and more. This derivative project does not provide enhanced functionality, but instead supports the parent project. As the core premise of that project is encapsulated in the parent Accumulo project, it is safe to omit these derivatives. After these operations were completed, 135 projects remained, allowing the researcher to manually pick projects for each quadrant. The proportions of these 135 projects that fall into each category of table 3 along with descriptive statistics per category are provided in table 6.

Table 6: Descriptive Statistics ASF Projects

Category	**# Projects per category**	**μ Innovation Score**	**μ Developer Score**	**μ Growth Rate**
Incremental, unsustainable	74	3.95	0.26	1.42
Incremental, sustainable	15	4.47	0.31	2.35
Radical, unsustainable	36	11.62	0.26	1.38
Radical, sustainable	10	10.23	0.33	2.15
Total	135	6.52	0.27	1.57

From the table, it becomes clear that the difference in Growth Rate between projects that are sustainable and unsustainable as well as the difference in Innovation Score between projects that are incremental and radical are large. This is expected given the construction of the operationalisations. As an important note, the author would like to clarify that when projects are addressed as "sustainable" or "unsustainable", this does not mean that "unsustainable projects" will not last. Instead, they are termed “unsustainable”because of their scores across the sustainability operationalisations, and should be treated as such.

Interestingly, while the Developer Scores differ between unsustainable and sustainable projects (similar to the Growth Rate), the order of magnitude is smaller. These numbers show that the projects within the different categories are sufficiently diverse, and hence, picking a representative selection of projects from these categorisations is feasible. Note that this table is thus (yet) not used to derive any preliminary conclusions about the projects, but it merely functions to show the reader how the projects are distributed over the categories.

In the end, the eventual choice for two projects per quadrant came down to selecting a popular project and a less popular project. This was done to analyse projects of different sizes (as popular projects are often larger than their less-popular counterparts) and the resulting dynamics that emerged from them. To determine whether a project was popular or not, the GitHub star rating was used. When a developer or user “stars” a certain project on GitHub, he or she saves it for later use and can easily find the project again[18]. Working with stars in this way provides an implicit indicator of a projects popularity, use, and the interest of developers to work on the project (Borges and Valente, 2018; Borges et al., 2016). While the choice to analyse popular and less popular projects might have influenced the constructions of the propositions in the attribute sections (e.g., “Incremental and sustainable projects that are popular have...”), the author still chose to make this distinction, as otherwise the selection of projects would have been too daunting. When choosing the projects for one category (say radical, sustainable), it was ensured that the projects were not of the same kind or tried to solve the same business problem. For example, if both chosen projects in a quadrant develop Data Base Management System solutions, then comparing the projects and finding differences among them might become more difficult. Table 7 provides the categories, the average number of stars per category, the chosen project for this study, and their respective stars. The average number of stars per category is displayed to show how the chosen projects differ in popularity in comparison to their respective categories. It also functions to highlight that these popularity metrics only make sense within the categories, as a popular project like IoTDB is popular in the category radical, sustainable but not in the category incremental, unsustainable (i.e., the number of stars for the IoTDB project is 3200, while the average number of stars for incremental and unsustainable projects is 4500).

Table 7: ASF Project Choice and Stars

Category	**μ # Stars within Category**	**Project**	**# Stars per Project**
Incremental, unsustainable	4500	Airflow	30000
		Cloudstack	1400
Incremental, sustainable	3100	Pulsar	12600
		Kyuubi	1600
Radical, unsustainable	6200	Skywalking	21700
		Helix	400
Radical, sustainable	2300	IoTDB	3200
		Eventmesh	1300

4.2 Data Sources and Terminology

Before diving into the sections that discuss and analyse the attributes from table 4, the author felt the need to provide information on two parts: The sources of data and the subsequent terminology that was used in the remainder of this study.

[18]https://docs.github.com/en/get-started/exploring-projects-on-github/saving-repositories-with-stars

4.2.1 Data Sources

This section functions as a point-of-reference for the attribute sections below. Much like the iterative way of working employed in the GT approach, the need to collect different sources of data arose to substantiate results that were already found, with data that was already gathered. Table 8 provides a final list of the data sources employed per attribute. The table further supports the implementation of triangulation in this study by supplementing the use of multiple methods per attribute (see the sections below) with multiple data sources per attribute, but only if it fitted the data at hand.

Table 8: Data Source per Attribute

Attribute	**Source**	**Clarification**	**Volume**
Nature of communication	github.com	The contents of the issue discussions were directly copied from the GitHub issue web-page for the respective projects.	1.207 comments
Work structure	github.com	The documentation on project releases was examined directly from the website for the respective projects.	235 releases
	Table 6	Table 6 from section 4.1 was referenced for the construction of one of the propositions.	-
Governance and leadership	github.com	Some of the documentation that discussed governance and leadership could be found on the github.com pages for the respective projects.	7 documents
	Project-specific documentation	Some of the documentation that discussed governance and leadership could be found on the websites of the respective projects.	16 documents
License type	github.com	The contents of the Apache 2.0 License were examined on the github.com license pages of the respective projects. As all projects listed the same license, it was only necessary to retrieve and read one such license page.	1 document (the Apache 2.0 license)

Continued on next page

Table 8: Data Source per Attribute — Continued

Attribute	Source	Clarification	Volume
	opensource.org	The other licenses (outside of the Apache 2.0 license) that are discussed in section 4.6 were examined through the open source initiative website.	2 documents (for GPL version 3 and LGPL)
Firm involvement	GitHub API	The graphs found in this section were constructed with data from the GitHub API.	1600 issue and pull request discussions
	apache.org	Documentation on sponsorship of the ASF was found and examined on their website.	3 documents (on sponsorship within the ASF)

4.2.2 Terminology

This terminology section serves to clarify some concepts to the reader before the attributes are tackled. While it would have been possible to clarify these concepts in their respective sections, the author deemed it best to present them here, as many of the following sections use the same terminology, and hence explaining them in an asynchronous or duplicate manner would not only unnecessarily elongate this studies contents, but also potentially confuse the reader. It is important to note that while some concepts that have been introduced before (like that of the "developer") will be re-formulated and split into smaller parts, the operationalisations of sections 2.1.2 and 2.2 remain valid, and the use of those operationalisations (for example in section 4.1) does not change.

First, it should be noted that the concepts and terminologies addressed in this section are true and legitimate in the context of the ASF, but different terminologies might apply to other firms. This contrast might be clearer for the ASF than for other firms, as the ASF is a non-profit foundation[19]. As all of the ASF's documentation on these concepts, terminologies and roles is found on their website, the author will use footnotes to reference this qualitative material wherever appropriate, and will do so in the same manner for the following attribute sections when such documentation is referenced.

"Users" refer to the individuals who use the (intermediary) results of the OSS projects. They might provide feedback on the software developed in the projects or file bug reports, but they are not active in the actual development of the project[20].

"Developers" are individuals that contribute to an OSS project regardless of the contents of that contribution, the roles that they fulfil, the responsibilities that they might have, or their authoritative rights. In the previous sections of this study as well as in the remaining sections, the term "developer" was and will be used when it is not necessary to make any more specific distinctions between different types of developers that exist. However, for some sections and discussions, such a distinction is necessary. "Contributors" contribute to the code of a project, track its documentation, provide suggestions for development, and participate in discussions[21]. They do not have any specific responsibilities or authoritative rights, so they cannot make changes to the code themselves. "Committers" are contributors who have

[19] https://www.apache.org/foundation/how-it-works.html
[20] https://www.apache.org/foundation/how-it-works.html; https://www.apache.org/foundation/governance/
[21] https://www.apache.org/foundation/how-it-works.html

push rights within a certain project (i.e, have the authority to make changes to the code)[22]. With these rights, they can make short-term decisions on the development of the project, but the permanency of those changes depends on the decisions of the "Project Management Committee (PMC) members". PMC members are individuals who are part of the PMC. These members are committers who collectively orchestrate the daily functioning of a project[23] and have full push rights (so, they make the decisions on which decisions are permanent). Collectively, they hold votes on community decisions (as no single PMC member holds the rights to any major decision) and decide on which contributors can become committers, and subsequently, PMC members. The selection criteria for becoming a committer or PMC member are based on "meritocracy" that is inherent in the Apache way of working[24]. In short, one must "prove his worth", often in implicit and tacit ways, to be eligible for committer or PMC member status[25]. The "PMC chair" is not the individual that leads the PMC (as no single PMC member has that authority) but rather functions as the link between the PMC and the Board of Directors.

The "Board of Directors" is the highest decision-making organ within the ASF (currently consisting of 9 members) and orchestrates the functioning of the foundation as a whole[26]. They do not lead the PMCs of the projects or make decisions for them (the PMCs are self-governing) but rather support them with infrastructure, resources, and sponsorships that they receive from other firms.

Finally, "ASF members" are committers or PMC members who have attained membership within the firm. Note that an ASF member is always either a committer or a PMC member, but committers or PMC members do not have to be ASF members. They annually decide on the composition of the Board of Directors, giving the foundation a "ruled by the masses" governance structure[27]. While many other different types of roles exist within the ASF (and even within the Board of Directors) such as treasurers, officers, vice presidents, administrators, etc., they are not of importance to this study.

4.3 Nature of Communication

4.3.1 Iterative Data Analysis

Analysing the nature of communication in the projects starts by defining the approach that was taken, which not only applies to this section and subsequent attribute but to all attribute sections. This approach involves analysing a subset of data, of which the findings are noted as constructs, formed by concepts and associations between those concepts by the author (also see figure 5 in section 4). One can also view these as "intermediate results". Then, other sources

[22] https://www.apache.org/foundation/how-it-works.html;
https://www.apache.org/foundation/governance/

[23] https://www.apache.org/foundation/how-it-works.html;
https://www.apache.org/foundation/governance/;
https://www.apache.org/foundation/governance/orgchart

[24] https://www.apache.org/foundation/how-it-works.html;
https://www.apache.org/theapacheway/index.html

[25] Note that this does not only include contributions to new and existing code, but can involve other activities as well.

[26] https://www.apache.org/foundation/governance/;
https://www.apache.org/foundation/governance/board.html;
https://www.apache.org/foundation/bylaws-plain-english.html;
https://www.apache.org/foundation/governance/members.html;
https://www.apache.org/foundation/governance/orgchart

[27] https://www.apache.org/foundation/how-it-works.html;
https://www.apache.org/foundation/governance/;
https://www.apache.org/foundation/governance/board.html;
https://www.apache.org/foundation/bylaws-plain-english.html;
https://www.apache.org/foundation/governance/members.html;
https://www.apache.org/foundation/governance/orgchart

of data are iteratively analysed, and the constructs that were formed before are either updated, supplemented with new constructs, or rejected until theoretical saturation is achieved. Finally, after theoretical saturation is achieved, can the constructs be translated into this study's propositions. In order to uphold a high degree of qualitative analysis, only if both projects within a category (for example Airflow and Cloudstack within the incremental, unsustainable category) show the same results (i.e., the same constructs guided by the concepts and associations that form them) will conclusions be derived for that category and subsequent propositions be made. This is also true for conclusions that are derived from "categorisations" of sustainable (or unsustainable) and incremental (or radical) classifications. It is done this way for all categories and for all attribute sections that follow. As a final note, it should be stated that the constructs are free-formed and a product of the author's interpretation of the material, which is inherent to the GT approach.

4.3.2 Chat-GPT

To gain an understanding of the nature of communication (which differs in informal and formal communication), the discussion between developers that is present in issues for a project on GitHub will be examined. Note that it thus does not matter if a developer is a contributor, committer, PMC member, etc., as the author is only interested in the nature of communication, not where that nature originated. Discussions on the GitHub platform present themselves in the form of comment threads (or comment "chains") which function as a mode of communication for developers, allowing them to express their thoughts about concepts and comments of other developers. Both issues and pull requests provide an opportunity for developers to have a discussion, but differences between issue discussions and pull request discussions exist. Pull requests, by their very nature, focus on getting a piece of code or some feature approved to the existing code base by a developer (or a small group of developers). The discussions on these requests are centred on the form and applicability of the code, thus being small in scope. In addition, reviewers (who are often committers or PMC members that test the form and applicability of the code) point out mistakes or points of improvement in the code to the developer(s) who submitted the pull request. The developer is then requested to improve the code until no mistakes remain and the pull request can be accepted into the code base. Although it does not have to be the case, this can result in a rather formal and rigid interaction. On the contrary, issue discussions are more free-form, allowing the participants of the discussions to talk about several topics. In addition, the interactions from reviewers to developers do not exist in these discussions as no pull requests are addressed, which can (but does not have to) result in a more diverse group of developers communicating among each other. For these reasons, it was chosen to analyse issue discussions in favour of pull request discussions.

By superficially scanning some long issue threads, the author quickly realised that analysing the nature of communication by relying on his own concepts and associations would be a challenging task. Although by its design, GT forms constructs based on the researchers interpretation of the material at hand, it is predominantly done by looking at the content of the material. However, when analysing the nature of communication, one is not interested in the content of the material but in the formality of that material. This makes it very difficult to derive consistent constructs that are not biased by the author's (changing) lexicon and world view (i.e., the concepts and associations that he derives), especially since the information in the constructs is context-dependent. To alleviate this problem, the recently introduced Chat-GPT[28] Artificial Intelligence (from now on addressed as "the AI") is employed to derive consistent notions of formality of the data at hand. Before explaining how the AI was used in this study and clarifying a couple of choices that were made in this process, one must first have a rudimentary understanding of how the AI functions. Being a language model, the AI can accept "prompts", which are questions written in plain English (i.e., the use of message compilation and decoding is not required, the AI understands human speech), and a host of restrictions and

[28] https://openai.com/blog/chatgpt

conditions can be applied to these questions if desired. As an applied example, consider the following prompt and the AI's response:

> *Prompt*: "Explain the concept of Open-Source Software to me in about 100 words."
>
> *Response*: "Open-source software refers to computer programs that are distributed along with their source code, allowing anyone to view, modify, and distribute the code. The open-source philosophy emphasizes transparency, collaboration, and community involvement. Unlike proprietary software, which is typically developed and controlled by a single entity, open-source software encourages decentralized development and collective improvement. This approach enables individuals and organizations to freely access, use, and customize the software according to their specific needs. By harnessing the power of collaboration and peer review, open-source software fosters innovation, promotes knowledge sharing, and often results in high-quality, reliable software solutions."

This example illustrates how the AI might be able to derive a notion of formality of a piece of text if asked the proper question(s).

4.3.3 Artificial Intelligence Restrictions

The example in the above section illustrates how one might go about using the AI to derive notions of formality of pieces of text. For example, if one wants to analyse the longest issue discussion chain of the Airflow project (#19891[29]), the comments that make up the discussion can be "fed" (i.e., given) to the AI, along with the question to rate the given data based on formality. However, it is not as simple as giving the AI one long string of the merged comments of a discussion thread in one go, as this is not possible due to the word-limit restrictions that the AI is built on, and would introduce a host of problems. To properly use the AI, several points must be taken into consideration and certain choices clarified. First, each comment in a discussion thread needs to be individually fed to the AI, as otherwise the formality of comment A might influence the formality of comment B. For the same reason, the discussions themselves should be separated (so the contents of Airflow's issue #19891 cannot influence the contents of Airflow's issue #13542). In addition, due to a comment feature inherent to the GitHub platform, merging all comment contents could result in a lot of duplicate text. For example, when @devinbost[30] commented the following in Pulsar's issue #6054:

> "I noticed that each topic lives on a single broker, which creates a single point of failure. Is there any interest in making topics higher availability? (I suppose it would be a workaround, but it might help prevent other issues)."

Developer @sijie had the option to repeat a part (or everything) of the above comment in his own response to give his response context, only address a specific part of the comment, point out mistakes, etc. @sijie chose here to repeat the full comment as part of his response, which looked something like this:

> *"I noticed that each topic lives on a single broker, which creates a single point of failure. Is there any interest in making topics higher availability? (I suppose it would be a workaround, but it might help prevent other issues)."*

[29]On GitHub, issues are indicated by a # sign followed by a sequential number, which not only provides a simple way to distinguish between different issues that exist but also makes clear in what order the issues emerge.

[30]The @ handle is used on GitHub by developers to address other developers by their usernames (i.e., by their @ handles), akin to how usernames work on the internet.

> "We (StreamNative) have been helping folks from Tencent at developing a feature called Read-Only brokers. It allows a topic can have multiple owners (one writeable owner and multiple readonly owners). It has been running on production for a while. They will contribute it back soon."

This could skew the formality in favour of pieces of text that get repeated a lot, so repeated text is not considered. Second, the punctuation of comments needs to stay intact (with regards to using commas, exclamation marks, periods etc., but also spelling, sentence structures and blank lines), as the AI is able to differentiate between these elements and is able to derive conclusions from them. For example, when prompted to rate the formality of a piece of text, responses like these were common for the AI:

> "However, the use of terms like "proper" and "unfortunately" adds a slightly informal tone."

> "The use of quotation marks around "dependent" packages and the smiley face emoticon also contribute to its informal tone."

What ties into this is that the AI is an English language model, and can only truly comprehend prompts that are given in English. As a result, the AI could not decipher other languages like Chinese, see the example below:

> *Prompt*: "Rate the formality of the following text: 大唐先一科技，由最初的0.9到目前的0.12，经历了多个版本的测试、反馈、调优、上线、升级，下一步将考虑单机到集群版的升级，大力支持国产软件的推进。"

> *Response*: "The provided text appears to be in Chinese. As a language model trained primarily on English text, I can provide a general assessment of formality based on visual cues and patterns.
>
> The text seems to be informal in nature. It includes numerals, abbreviations, and technical terms, which can suggest a more specialized or informal context. However, without more context or knowledge of specific linguistic nuances in Chinese, it is challenging to provide a precise formality rating."

The AI is able to make a somewhat educated guess by picking up on abbreviations used in the sentence and its use of numerals, but explicitly states that without more knowledge of the language, it is unable to derive definitive conclusions. One option that the author considered was to translate the text first (for example using an AI-supported translation tool like DeepL[31]) and then give it to the AI, but this might introduce translation inaccuracies, as it could for example lose linguistic nuances.

Third, bots can comment as well, as their automatic behaviour executes once a certain condition is met. For example, when a developer opens his first issue discussion in the Airflow project, a bot automatically responds with the following: "Thanks for opening your first issue here! Be sure to follow the issue template[32]!" These responses by bots never lead to a direct discussion because the human developers present in the discussion thread do not respond to these messages. While one could argue that the response by the bot actually provides some information on the notion of formality in a text (as it is pre-programmed to respond in a certain way), it does not organically arrive from the discussion. Thus, responses by bots are excluded (i.e., the AI is not given the comments that are made by bots). Fourth, the initial comment that opened the issue discussion is not included. These messages often contain a lot of task-related information, which is probably interesting for the contents of the discussion but not so much for its notion

[31] https://www.deepl.com/en/translator

[32] An issue template provides guidelines on how issues should be opened and what information should be included. Templates like these are often project-specific.

of formality. In addition, one can argue that this first message merely "opens" the discussion, and at that point, the discussion has not yet truly started. For these reasons, the first comment of a discussion is not considered. Finally, GitHub comments allow developers to include dedicated environments that display technical information (like error logs and python scripts for example). These environments were excluded from the prompts to the AI for two reasons: One, by leaving this information in, some prompts would become too large for the AI to handle (recall the word-limit restrictions mentioned earlier), and two, as the author thinks that while the inclusion of technical information tells something about the content of a comment and the expertise of a developer, it does not say anything about his or her formality when communicating. The AI seems to agree, see the following snippet of a response as an example:

> "While it discusses technical details and includes specific references to comments and suggestions, the language used is conversational and lacks a strictly formal structure."

In addition, comments that only consisted of technical information (e.g., containing error logs and code dumps) were not given to the AI. Note here that technical information that appeared in-text (so, within lines of text and outside the dedicated environments) was still included, as removing these elements from the sentences could confuse the AI's interpretation of sentence structure and punctuation, also see the point made earlier.

With these explanations in place, the author was able to use the AI and extract the notions of formality that could be found in the comments of the three longest closed issue discussions per project. It was chosen to analyse the three longest discussion threads (as opposed to only the first one) to see if the derived notion of formality was consistent across the project as a whole. While the author would have preferred to include more issue discussions, the manual labour required to hand-feed the AI every individual comment prevented the author from doing so. In addition, the AI not only restricts the number of words that can be given in a prompt but also the number of prompts it accepts per hour, further slowing down the process[33]. The choice to analyse the longest discussions was made because these discussions provide the most information about how developers communicate with each other (as opposed to shorter discussions). In addition, only discussions that were closed were examined, as these discussions were (generally) finished when closed by the original poster of the issue or another issue participant, providing consistency across issues.

Before the step-by-step process that was used to work with the AI can be outlined, some exceptions that were made for two projects need to be clarified. The first exception was for the Helix project, as the longest issue discussion threads for that project only consisted of the comments of bots, and as specified before, these responses are not considered. In addition, many other long issue discussion threads that were not led by bots were used by singular developers to keep track of their progress on an issue. As these are not really discussions between developers (but rather, between a singular developer himself), they could not be given to the AI. Instead, the author selected the three longest issue discussion threads (that actually involved comments between and directed toward different developers) by hand (#609; #349; #1336). The second exception was made for the IoTDB project, as several of the longer discussions were primarily held in Chinese. As shown before, the AI is unable to interpret prompts that are not in English, and translating the prompt beforehand could introduce problems with linguistic nuances and accuracy. Instead, the author opted to skip a couple of the longest issue discussions of this project (as these were predominantly, or even fully, in Chinese) and instead analyse the 7th, 10th and 13th discussions (#2124, #2217, #9326) as these discussions were in English.

Finally, the process used to interact with the AI is given below:

- Step 1: Start a new chat[34] (recall the confounding of formality).

[33] The author also inquired if the more advanced version of Chat-GPT (called Chat-GPT PLUS, also see https://openai.com/blog/chatgpt) did not have such a rate limit, but such information could not be found. It was thus chosen to employ the free, standard version of Chat-GPT and work with its limitations.

[34] Starting a "new chat" is indicating that you want to initiate a new chain of prompts and responses.

- Step 2: Copy the contents of the first (usable) comment (addressed as [comment_content]) of an issue discussion and give the AI the following prompt—“Rate the formality of the following text: [comment_content]”.
- Step 3: Repeat Step 2 until all (usable) comments are given to the AI and the AI has responded to all of them[35].
- Step 4: As a last prompt, give the AI the following—“Based on the answers to the prompts in this chat, provide a statement on the weighted notion of formality.”
- Step 5: Record the response of the last prompt given in Step 4.

4.3.4 Formality

The results of the process outlined in the previous section, whereby each of the three issue discussions per project is given a final response by the AI (see Step 4 and 5), can be found in table 16 in Appendix A. Before these results are examined, one final comment needs to be made about the workings of the AI. After the AI responds to a given prompt, it provides the option (in the form of a button) to “regenerate” the response that is given. This usually means that the same answer will be given but worded differently. However, sometimes it can occur that the AI derives a different conclusion when a response is regenerated. To account for these (potential) internal inconsistencies, each final response from the AI (so the answer to the prompt specified in Step 4) was regenerated 5 times to see if any different results arose from the initial, first answer that is presented in table 16. Fortunately, the AI remained consistent in answering all final prompts, only changing the wording but not its contents.

Analysing the AI responses for the Airflow and Cloudstack projects, it became clear that there was no consistency among this category (i.e., within the category of incremental, unsustainable). While the Airflow project showed internal consistency (i.e., all three responses from the AI declared the nature of communication for this project to lean to the informal side), the responses for the Cloudstack project swayed from informal to formal. This missing consistency, however, led to the author’s suspicion that the nature of communication for projects might be different depending on which specific project is examined. Next, after analysing the Pulsar and Kyuubi projects, the same patterns were observed. The Pulsar project showed internal consistency, while the Kyuubi project’s responses differed (one leaning toward a more formal nature of communication and the other two toward a more informal nature of communication). Spurred on by these findings, all remaining projects were analysed as well (i.e., the Skywalking, Helix, IoTDB and Eventmesh projects), whereby no consistency among categories could be found, further confirming the author’s ideas about the different natures of communication for each project. To confirm these ideas, the author decided to analyse the longest discussion thread of the Airflow project (#19891) by hand. As the author examined the comments and derived conclusions from them based on his own understanding of formality, it was observed that the formality employed by different developers was inconsistent. This seemed interesting, as differing levels of formality employed by developers within the same issue discussions might indicate that the nature of communication not only differs based on the specific project looked at, but also by which developers participate in these discussions. As an example, consider the following comment by developer @potiuk:

> “BTW. After few biggish (common) fixes I plan to re-generate the list and check where we are later today :)”

This includes the use of abbreviations (BTW instead of By The Way), casual language (e.g., biggish) and an emoticon (a smiley face), making this piece of text fairly informal (based on the author’s understanding). In contrast, other developers (like @josh-fell) use a more direct and formal language, like so:

[35] One cannot give the AI a new prompt before the AI finishes its answer to the previous prompt.

> "You could use the # type: ignore[attr-defined] marker so Mypy ignores that specific error."

Which the author considers to be more formal due to the absence of abbreviations and emoticons, as well as employing a more serious tone and providing a direct, although professional and non-intrusive, suggestion for ignoring a specific error. These examples illustrate the varying degrees of formality in which developers respond. In addition, developers do not have to employ the same formality throughout the entire discussion. For example, developer @potiuk replied more directly and formally when the situation demanded so:

> "I think I have all necessary changes for Google: #20358, #20329, #20234"

This shows that the nature of communication not only differs from developer to developer but also can differ for a single developer, depending on the context of the discussion and the situations that they find themselves in. Based on this information, the following proposition was made:

> **Proposition 1**: The nature of communication varies from formal to informal based on the specific developer, context, and project examined.

Interestingly, not a single project could be categorised as having a ubiquitous formal style of communication, which was understood here as internal consistency for a project in terms of the AI's responses, where all three of them would indicate a formal nature of communication. After giving this some thought, the author concluded that this was probably due to three things: One, because of the core ideas of OSS—Anybody (as long as they have the required knowledge) can contribute, whereby no hierarchical structure of command is present, as it could slow down the development of the project and make decisions rigid and inflexible. Two, the design of discussions on the GitHub platform only allows developers to address each other by their user handles (as they (probably) do not know each other's real names), which always leads to a direct and informal nature of communication. Finally, this more informal way of communication may be cultural to the ASF and its non-profit nature. Perhaps if a for-profit firm was analysed, different outcomes would have been found.

4.4 Work Structure

4.4.1 Superposition and Co-work

Collaborative efforts within the world of OSS can be typified in two ways: Incremental and individual (called superposition) and collaborative and co-work. Before diving into the projects and examining these different efforts, a comment on how collaborative work on platforms like GitHub functions needs to be made. By design, GitHub (and similar platforms) allow developers to collaborate with each other and collectively develop code. As such, if a project is built by more than one developer (i.e., it is not an individually led project anymore, which is the case for all projects examined in this study) there will always be an inherent collaborative aspect to it, as the developers need to communicate with each other and work together to reach desired results. For this reason, in this study, a "truly incremental and individually developed project" does not exist. That said, projects can be typified as more incremental and individual or more co-work and collaborative. To be able to make such a distinction, an approach to analyse the work structures of projects needs to be clarified.

In their 2014 paper, Howison and Crowston (2014) define the difference between a superpositioned work structure and a collaborative or co-work work structure. Superposition (also known as an incremental or individual work structure) is defined as a work structure whereby developers individually write code and add it to an existing code base in a step-by-step manner, effectively creating layers of work on top of each other. Instead of tackling complex tasks that require the involvement of many developers, superposition-oriented developers wait until a large group of individual developers (who individually add their layers of code to the project) provide the code necessary to take on a complex task. Then, as the complex task gets executed, it might provide a further basis for the tackling of new incremental tasks until another complex task can be handled. In contrast, co-work[36] focuses on the joint development of code, providing modular blocks that multiple developers worked on. Medappa and Srivastava (2019) further examined the nature of these work structures and specifically focus on superpositioned development, as it has been argued to be the main approach to working within the FLOSS environment (Howison and Crowston, 2014). They define their construct (i.e., their degree of superposition) as "the ratio of the total number of versions of the FLOSS project to the total number of individual task contributions to the project" (p. 773). In this way, the more individual developers co-work on a released version of the project, the lower the ratio of superposition becomes, and vice versa.

4.4.2 Release Versions

The approach taken by Medappa and Srivastava (2019) cannot be employed in this study because the information that disclosed the individual task contributors for each version of the projects could not be gathered by the author. However, it does provide a basis for how superposition (and consequently co-work) can be analysed in this study in a qualitative manner. Instead of examining the individual task contributions to a project, the release history (i.e., the different versions) of a project can be analysed to see if a project leans toward superpositioned work or co-work. More specifically, if a project has many smaller releases (called minor releases) compared to its number of big releases (called major releases), one could say that the work structure of that project is akin to a superpositioned work structure. Similarly, if a project has a few minor releases compared to their number of major releases, one could say that the work structure of that project is similar to a co-work work structure. Even in the absence of individual task contributions, which makes this approach lose some explanatory power, the frequency of major and minor releases can still provide insights into the work structure of projects.

To this end, the author analysed the release history of the projects through their release information found on GitHub, the results of which can be found in table 9. It shows the total number of releases per project, as well as the number of major and minor releases, whereby the percentages indicate the proportion of major (or minor) releases compared to the total number of releases per project. Before these numbers can be analysed, it needs to be clarified what is meant by a major and a minor release in this study. The definitions for these concepts (i.e., major and minor releases) have two components: Version numbers and release contents. First, if structured properly (which was the case for each of the projects analysed in this study), the version number of the release indicates what type of release it entails. As an example, the version numbers of Airflow's releases can be looked at: Version 1.4 came first, then 1.5, and then 1.5.1. As 1.5.1 is the 0.1 version of the 1.5 release, it can be seen as a minor release. It likely only provides small updates like bug fixes and patches to security issues. In contrast, 1.4 and 1.5 are major releases, as the jump in numbers from 4 to 5 indicates that the 1.5 release not merely builds on the 1.4 release but provides something substantially new or improves on important features. Otherwise, a 1.4.1 release would have been created. To validate these assumptions, the author also examined the contents of the releases, confirming or denying whether the version number was indeed indicative of a major or minor release. Note here that the size of a release (understood here as the amount of content that it covers) is not indicative of a major or minor release, as this differs for projects based on the size of the project. For two projects an exception needed to be made. The first was for the Airflow project, as its release history not only detailed the releases of the main project but also those of two side-projects (Helm Chart and

[36]From now on, collaborative and co-work will be addressed as co-work only, as all work on an OSS project (regardless of focusing on a superposition or co-work work structure) is inherently collaborative. Thus, "collaborative" as a term to indicate co-work here will be avoided.

Upgrade Check). As these side-projects were separate from the main project (both in content and release order), they were not included in the numbers in table 9. Second, a couple of Skywalking's first versions were released when the project was not yet part of the ASF (clearly indicated by the "Not Apache release" declaration in the version titles) and hence are not considered. Finally, the first version of a project is seen as a major release as it details the project's initiation, and pre-releases are not taken into consideration as they are seen as predecessors to actual releases.

Table 9: Major and Minor Releases ASF Projects

Category	Project	# Total Releases	# Major Releases	# Minor Releases
Incremental, unsustainable	Airflow	47	12 (26%)	35 (74%)
	Cloudstack	20	8 (40%)	12 (60%)
Incremental, sustainable	Pulsar	67	22 (33%)	45 (67%)
	Kyuubi	29	14 (48%)	15 (52%)
Radical, unsustainable	Skywalking	30	7 (23%)	23 (77%)
	Helix	6	3 (50%)	3 (50%)
Radical, sustainable	IoTDB	26	8 (31%)	18 (69%)
	Eventmesh	10	9 (90%)	1 (10%)

Analysing the table it becomes clear that no constructs can be derived for the different quadrants as there is no consistency to be found for the different categories. In addition, except for the Helix and Eventmesh projects, all projects have fewer major releases than minor releases. This is an unusual finding, as the author expected only incremental projects which, by their definition in this study, operate in increments to commits and file changes to favour a more superpositioned working structure (i.e., having more minor than major releases). Instead, both incremental and radical projects (except for the Eventmesh project) favoured a superpositioned working structure, or in the case of the Helix project, equal weight was given to both. To check the validity of these findings, the temporal aspect of a project's superposition (i.e., its maturity) as a potential explanatory factor was examined. This was also raised as a future point of inquiry in the Medappa and Srivastava (2019) study. When examining the average age in years of all projects in this study's pool (so the N = 135 found in table 6 in section 4.1), the following values were found for each category: Incremental, unsustainable: 13.01; Incremental, sustainable: 7.87; Radical, unsustainable: 12.31; Radical, sustainable: 7.5. While both incremental and radical projects have high and low values for the average age of a project, another interesting finding can be made: Projects that are sustainable have (on average) a lower value for a projects age (7.87 & 7.5) than projects that are unsustainable (13.01 & 12.31). This can be explained by the higher average Developer Scores and Growth Rates of sustainable projects (see table 6) and poses the following idea: When a project matures, its work structure shifts from co-work to superpositioned work. Note here that it makes sense that the unsustainable projects are more mature, as unsustainability in this study is defined as a combination of a low Developer Score (i.e., a few core developers and several peripheral developers) and a low Growth Rate (the initial "burst" of developer growth at the start of a projects life-span is already over). Based on this finding, the following proposition can be made:

Proposition 2: The work structure of projects transitions from co-work to superpositioned over time.

In addition to examining the maturity for all projects (which led to the construction of the above proposition), the author realised that table 9 and the conclusions derived from that table could be further validated by controlling for another temporal aspect: Release dates. Not only can the cumulative number of (minor and major) releases be analysed for each project but it can be further controlled by examining how many releases occur each year, irrespective of whether these concerned major or minor releases. For example, if incremental projects have more releases each year than radical projects (on average), it can indicate that incremental projects favour a more superpositioned working structure. To this end, table 10 was constructed. It details the average number of releases per year per project. As with table 9, the same conditions apply.

Table 10: Yearly Releases ASF Projects

Category	Project	μ # Releases per Year
Incremental, unsustainable	Airflow	6.71
	Cloudstack	3.33
Incremental, sustainable	Pulsar	8.38
	Kyuubi	4.67
Radical, unsustainable	Skywalking	5
	Helix	1.5
Radical, sustainable	IoTDB	5.2
	Eventmesh	2.5

From the table, it can be garnered that on average, incremental projects have 5.77 releases per year while radical projects have 3.55. This leads the author to believe that incremental projects favour a more superpositioned working structure and radical projects a more co-work working structure, which is reflected in the following propositions:

Proposition 3: The work structure of incremental projects can be typified as superpositioned.

Proposition 4: The work structure of radical projects can be typified as co-work.

4.5 Governance and Leadership

Governance and leadership are not mutually exclusive concepts, as leadership sets the direction of a project and decides what will happen, while governance is the enforcement of that direction[37]. After some initial examination by the author of the nature of governance and leadership in the context of OSS projects, it became clear that the two concepts are tightly intertwined. The central premise of OSS is that (within reason) anyone can contribute and that there is no singular entity who decides what will be done and how those decisions are enforced. This is in line with the way of working of the ASF and explained terminology in section 4.2.2. More specifically, recall that how contributors become eligible for committer or PMC member status is to "prove their worth" through meritocracy—Favour is gained by working hard and contributing to the project. However, this does not mean that governance and leadership are absent in OSS projects.

4.5.1 Democracy, Transparency and Structure

To unravel the governance and leadership structures present in projects, O'Mahony and Bechky (2008) and O'Mahony and Ferraro (2007) presented two works on governance and leadership in OSS communities and their interactions among themselves as well as with firms. The former investigated how OSS communities and firms create boundary organisations that mediate the conflicting interests of the two parties, while the latter (among other things) examined the emergence and change in governance and leadership structure for an OSS project over time. Both studies conducted interviews with the involved parties to gather information. While it is not possible to do the same in this study, both works (in their methodology and subsequent analysis) did highlight important aspects of governance and leadership that can be considered and examined here. More specifically, three areas of interest stand out: Democracy,

[37] See for example the report by the NFER (https://www.nfer.ac.uk/media/1939/lgg01.pdf).

transparency, and structure. First, in the works of O'Mahony and Bechky (2008) and O'Mahony and Ferraro (2007), democracy is discussed: How much power is in the hands of a singular entity (i.e., autocracy or centralised governance and leadership) versus multiple entities (i.e., democracy or decentralised governance and leadership). O'Mahony and Bechky (2008) mainly examined the power struggles that arose from cooperation between firms (which was seen more as the singular entity) and OSS communities (which were seen more as the multiple entities) in search of shared interests. Similarly, O'Mahony and Ferraro (2007) showed how the control of a singular entity (who was also a developer of the project, so not a firm-affiliated individual) eroded over time in favour of more (dispersed) control by the other developers on the project. Second, transparency describes the willingness of firms and OSS communities to disclose information about their ideas and intentions toward governance and leadership, as discussed in the study of O'Mahony and Bechky (2008). Third, structure describes the breadth and depth of the agreements made between parties about governance and leadership. Often, in the study of O'Mahony and Bechky (2008), parties had to adjust and reform their agreements with each other based on a new understanding of the roles that different entities played and the influence that they would have over the projects. In a similar vein, the community of developers in the study of O'Mahony and Ferraro (2007) had to update their control over the leadership and governance of the project when problems with regard to autonomy and decision making arose. Note thus that transparency and structure are often related but remain distinct from each other, and range from low to high (e.g., low structure to high structure).

4.5.2 Project Documentation

The three areas of interest (i.e., democracy, transparency and structure) specified in section 4.5.1 were examined for the projects in this study. Instead of relying on interviews, the documentation that each project provided on governance and leadership was examined. Note that these documents differ from the more general documentation that was provided in section 4.2.2, as the former provides project-specific information. The location of the relevant documentation differs from project to project, as some projects host their entire documentation on GitHub (e.g., Airflow) while others provide links to external documentation on the site of that project (e.g., Kyuubi) or use a combination of both (e.g., Cloudstack). Before examining this documentation on the areas of interest, it should be made clear how these documents are looked at and judged. If the project-specific documentation for example does not mention the PMC, the difference between contributors and committers, or the use of formal (i.e., structured) procedures, this does not mean that these concepts (i.e., these roles and guidelines) are not present in the projects. Instead, the creators of the documentation did not deem it necessary to explain or mention these things, or (according to the author's reasoning) implicitly assume that the general documentation that is provided on the ASF website that covers these concepts is known to anyone interested in the ASF and its workings. Therefore, instead of using this project-specific documentation to make claims about the presence or absence of certain concepts in a project (which is not reasonable as all are always present in each project, see https://www.apache.org/foundation/how-it-works.html), it is used to show how and where emphasis has been placed, and thus how democracy, transparency, and structure differ.

Starting with the documentation on incremental and unsustainable projects (so, Airflow and Cloudstack), it immediately became clear that these two projects differed in both their transparency and structure of documentation. For once, the Airflow project provided documentation on the different roles within a project [38] (i.e., transparency is high) and how those roles are filled[39] (i.e., structure is high). The documentation on the project and its functioning is extensive, and the creators of the documentation (who, in case of the Airflow project, are committers and PMC members) uphold a high standard of quality in everything that they do and make. For example, see their statement on testing providers for the project: “We have high requirements when it comes to testing the community managed providers.

[38] https://github.com/apache/airflow/blob/main/CONTRIBUTING.rst#id61
[39] https://github.com/apache/airflow/blob/main/COMMITTERS.rst

We have to be sure that we have enough coverage and ways to tests for regressions before the community accepts such providers."[40]. Similarly, they have clear views on how they want to see things getting done, take for example the use of pre-commit hooks when testing code: "Pre-commit hooks help speed up your local development cycle and place less burden on the CI infrastructure. Consider installing the pre-commit hooks as a necessary prerequisite."[41]. These illustrations highlight the culture that permeates the Airflow project and its different roles. It places emphasis (when compared to other projects) on standards and ways of working, which is seen by the author as having high structure and transparency. In addition, the two illustrations given before (on testing and pre-commit hooks), along with strict and extensive guidelines on how to become a committer and PMC member[42], lead the author to believe that the Airflow project is more autocratic, possibly led by a smaller group of strict PMC members.

Unlike the documentation on the Airflow project, the documentation on the Cloudstack project was much more compact and focused on a central component: The process of pull request and release policies. In these documents, explicit and detailed information is given on how a non-committer (i.e., a contributor) can use pull requests within the project to contribute to the code[43], as well as how merging of code and releases should be handled[44]. Interestingly, very little to no information about the actual governance structure was given (so transparency was low), and as indirectly mentioned before, there were no mentions of a PMC group. Only two pieces of information could be found on the decision-making process of this project. First, in the document that specifies how non-committers can contribute[45], an implicit reference is made to committers, showing that (at least) the distinction between contributors and committers is evident. Second, on the documentation of mailing lists, the following was found: "While the project has several communications channels, the mailing lists are the most active and the official channels for making decisions about the project itself. If it didn't happen on a mailing list, it didn't happen."[46]. These two points show that there is a distinction between roles (i.e., contributor vs. committer) and a couple sources of discussion and decision making (the mailing lists), which highlights that there is at least some form of structure present, even in the absence of other information about governance and leadership (at least that the author is aware of). It highlights that while little emphasis is given to the documentation and the explanation of roles and governance structures (in terms of volume), it does provide adequate structure, which is seen by the author as an indicator for the project employing a higher structure and leaning more toward autocracy. As both the Airflow and Cloudstack projects employed high structure and autocracy, the following proposition can be made:

> **Proposition 5**: The governance and leadership of incremental and unsustainable projects are autocratic and employ a high structure.

Like the Cloudstack project, Pulsar's documentation on contributions only extended itself to a page on their website[47] and one document on GitHub[48], indicating low transparency. Essentially, anyone can contribute to the project and does not need formal permission for making small changes (e.g., bug fixing, changing documentation, removing typo's). Only when proposing new features and changes to the APIs, a Pulsar Improvement Proposal (PIP) is required. This is a more formal document that communicates the ideas and reasoning for the change in a standardised way, allowing such suggestions to be handled in a standardised manner. While the creation and submission of such a PIP does

[40] https://github.com/apache/airflow/blob/main/CONTRIBUTING.rst#id61
[41] https://github.com/apache/airflow/blob/main/STATIC_CODE_CHECKS.rst
[42] https://github.com/apache/airflow/blob/main/COMMITTERS.rst
[43] https://github.com/apache/cloudstack/blob/main/CONTRIBUTING.md
[44] https://cwiki.apache.org/confluence/display/CLOUDSTACK/Release+principles+for+Apache+CloudStack+4.6+and+up
[45] https://github.com/apache/cloudstack/blob/main/CONTRIBUTING.md
[46] https://cloudstack.apache.org/mailing-lists.html
[47] https://pulsar.apache.org/contribute/
[48] https://github.com/apache/pulsar/blob/master/wiki/proposals/PIP.md

not guarantee the approval of its contents, “any person willing to contribute to the Apache Pulsar project is welcome to create a PIP”[49]. The PIP gets approved or rejected by the individuals who are in the dev@pulsar.apache.org mailing list, whereby dedicated PMC members hold binding votes. Due to the limited documentation on this, the author was unable to determine who the members of the mailing list or the PMC were, and neither could information be found on how to become part of these groups. However, the documentation did state that “everyone is welcome to vote on the proposal”[50], leading the author to believe that although not binding, a majority of votes in favour or against a PIP could sway or influence the decisions of the developers on the mailing list or the PMC members. This was actually very similar to the documentation provided by the Airflow project[51], and is thus seen by the author as having a high structure, even when transparency was low. In addition, the formality of the PIP procedure in combination with the focus on that particular procedure leads the author to believe that the Pulsar project is more autocratic. On the other hand, Kyuubi’s documentation (both on GitHub[52] as on their website[53]) provides even fewer information when compared to the other (already examined) projects, only stating that anybody can become a committer or PMC member (i.e., transparency is low). In addition, it gives a short statement on how there are no formal requirements for becoming a committer or PMC member. Thus, the only documentation that is present explicitly places emphasis on the divisions of these roles even in the absence of formal requirements to attain these roles, indicating a relatively low structure. For this reason and the fact that no other documentation is provided, the author believed the project is more democratic. Comparing the Pulsar and Kyuubi projects, the following proposition can be made:

> **Proposition 6**: The governance and leadership of incremental and sustainable projects employ low transparency.

The Skywalking and Helix projects were examined next. The Skywalking project only provides some short documentation on how to contribute[54], and is mostly concerned with how new contributors should set up their development environments, as well as providing a small snippet of insight into the workings of their PMC group[55]. So transparency is low. Other than that, it only references the general code of conduct on the general ASF website[56], indicating that its structure is low. As the only documentation on the PMC is very short, the author is inclined to typify this project as more democratic. In a (somewhat) similar vein, the Helix project provides little documentation, with its longest contribution being how new contributors should get started[57], indicating low transparency. Interestingly, while they do provide an extensive list of their current committers and PMC members[58], all other documentation (on issue management, mailing lists, integration etc.) is very concise and short, so structure is low. This specific focus on the current committers and PMC members does lead the author to typify this project as more autocratic. Comparing the two projects, the following proposition can be made:

> **Proposition 7**: The governance and leadership of radical and unsustainable projects employ low transparency and structure.

[49] https://github.com/apache/pulsar/blob/master/wiki/proposals/PIP.md
[50] https://github.com/apache/pulsar/blob/master/wiki/proposals/PIP.md
[51] https://github.com/apache/airflow/blob/main/CONTRIBUTING.rst#id61
[52] https://github.com/apache/kyuubi/blob/master/CONTRIBUTING.md
[53] https://kyuubi.apache.org/become_committer.html;
https://kyuubi.apache.org/become_pmc_member.html
[54] https://github.com/apache/skywalking/blob/master/CONTRIBUTING.md
[55] https://github.com/apache/skywalking/blob/master/docs/en/guides/README.md
[56] https://www.apache.org/foundation/policies/conduct
[57] https://cwiki.apache.org/confluence/display/HELIX/Contributor+Workflow
[58] https://helix.apache.org/team-list.html

Finally, analysing the IoTDB and Eventmesh projects, similar constructs were found as those that were analysed before. The IoTDB project's documentation is extensive (so transparency is high), combining elements of the previously analysed projects. Like the Airflow project, it extensively documents the different roles (contributor, committer, PMC member) within a project[59] and a general way of project functioning[60]. From the Cloudstack project, it mirrored the extensive documentation on the process of pull requests and how to contribute code as a beginning contributor[61], and finally, as in the Kyuubi and Airflow projects, it provided information on how to become a committer or PMC member[62]. These elements show a high structure, along with a more autocratic leadership style. On the contrary, the Eventmesh project only provides a relatively small amount of documentation on how to set up your development environment and how to contribute[63] (i.e., transparency is low) in addition to specifying how the mail lists work and where to subscribe [64], without going into too much detail, leading the author to classify its structure as low. In addition, it provided almost no documentation on actual governance and leadership, leaning more to democracy. As the IoTDB and Eventmesh projects differed in their democracy, transparency, and structure, no propositions could be made for this category (i.e., radical, sustainable).

4.6 License Type

At the beginning of section 1, OSS was defined as follows: "Open-Source Software projects consist of source code... This code is distributed to the public, allowing free use, modification and further distribution of said code, conditional on licensing agreements". This text already made it clear that while OSS can be freely used and further distributed among anyone, the type of license that is tied to it can place differing restrictions on that use and distribution. Choosing the right license for a project is no trivial task, as this choice can influence the success and outcomes of a project (Stewart, Ammeter, et al., 2006; Sen et al., 2011), determine its business potential (Lindman et al., 2010), and indicate the number of developers and their activity on the project (J. Colazo and Fang, 2009; J. A. Colazo et al., 2005). In this study, due to the choice of firm made in section 4.1, all examined projects operate under the same license, which originates from the ASF: The Apache 2.0 license. To understand the workings of this license and other licenses, one must first have an understanding of the different types of licenses that are being used within the OSS landscape. While there are more than 80 different OSS licenses in use today[65], they all fall under one of 3 categories: Weak copy-left, strong copy-left, and permissive. Note here that copyright licenses (which are often employed in fields of music, movies, books etc.) do not apply to OSS projects, as they would hinder the very core premise of OSS: Going open source. In addition, outdated or deprecated versions of licenses (like the Apache 1.0 license) will not be examined.

Weak copy-left and strong copy-left licenses share the same core premise but execute it differently. Strong copy-left licenses (like GPL version 3) state that if an entity (for example an individual or a firm) uses code from another entity that has a strong copy-left license tied to it, the former entity is allowed to make any modifications and changes to that code, and use it in its own code, under two conditions. First, this new original source code that uses elements from the strong copy-left code that it was derived from needs to be made public, just like the original strong copy-left code. Second, the newly released code must be released under the same strong copy-left license. These conditions

[59] https://iotdb.apache.org/Community/About.html;
https://iotdb.apache.org/Development/Committer.html

[60] https://iotdb.apache.org/Development/ContributeGuide.html

[61] https://iotdb.apache.org/Development/HowToCommit.html;
https://iotdb.apache.org/Development/HowtoContributeCode.html

[62] https://iotdb.apache.org/Community/About.html#_1-roles;
https://iotdb.apache.org/Development/Committer.html

[63] https://eventmesh.apache.org/community/contribute/contribute/

[64] https://eventmesh.apache.org/community/how-to-subscribe

[65] https://opensource.org/licenses/

make the strong copy-left license the strictest of all OSS licenses. Weak copy-left licenses (like LGPL) work similarly, but if an entity merely uses source code from a weak copy-left licensed code (i.e., without modifying it) and keeps it in a separate file from the new source code that depends on it, they can release the new source code (and the separate weak copy-left license code) under any new license of choice, making it less restrictive than a strong copy-left license. Finally, permissive licenses (like The Apache 2.0 license) are the least strict. While they are not exempt from license-specific conditions, they generally allow an entity to modify and use permissive code and distribute it further (under a new type of license if desired), so long as a notice (i.e., a copy) of the original permissive license is included. Naturally, with these 3 license types, the following question arises: Which license type can best be chosen for OSS projects? In other words, is a more restrictive (i.e., akin to a strong and weak copy-left) license better, or a less restrictive (i.e., akin to a permissive) one? The answer to this question depends on context, the measure of success employed, and the type of product (i.e., the project) that is tied to it. For example, Stewart, Ammeter, et al. (2006) found that non-restrictive licenses are best employed to attract user interest (with user interest being their measure of success), while J. A. Colazo et al. (2005) found that more restrictive licenses were better for attaining success when looking at developer membership (i.e., what is the number of developers working on a project) and developer productivity. Another example is the work of Sen et al. (2011), who argue that non-restrictive licenses will dominate restrictive licenses, but only "when a large effort is required to develop derivative software[66]" (p. 199). As this study does not employ measures of success (recall that the societal LOA was removed in section 3.2) but merely looks at factors that influence that success, using an approach as specified in the works of Stewart, Ammeter, et al. (2006) or J. A. Colazo et al. (2005) (that uses regression analysis) is not possible. In addition, analysing the licenses and their influence on project success in a more qualitative manner (for example by taking interviews like done in the work of Lindman et al., 2010) is not possible in this study due to time and resource constraints. As such, no propositions derived from qualitative data can be made here. While the author could opt for forming some preliminary hypothesis based on his own understanding of the licenses and the contents thereof which can be refined, examined and validated in further research, he chooses not to, as the formation of such hypothesis justify the set-up and execution of a separate study which will not be done here. In addition, note that the construction of propositions and preliminary hypotheses would have been difficult due to all chosen projects employing the same Apache 2.0 license, making comparisons within and between quadrants meaningless. Yet, the author still opts for the inclusion and mentioning of this attribute in the study, as the aforementioned studies in this section (and the studies provided in table 4 in section 3.3) make it clear that examining the structures of and influence on OSS project success while looking at licensing is important.

4.7 Firm Involvement

As stated in table 4 in section 3.3, firm involvement in a project originates from two sources: The firm that owns the projects (in this case, the ASF) and external firms that sponsor the projects, either by directly subsidising the ASF or paying employees to work on the projects. As such, this section chooses to separate the two, analysing them apart from each other.

4.7.1 Corporate Influence

Being a non-profit foundation and having self-regulated projects (see section 4.2), at first glance there is not much firm involvement and influence from "the top layer" by the ASF. Indeed, the documents specified in sections 4.2.2 and 4.5 indicate as much, and of the three management models specified by Capra et al. (2011), the ASF is categorised by its supportive nature. However, this does not mean that the ASF has no influence on the workings of its projects. The

[66] Derivative software is used as a term here to indicate the new product, i.e., the newly released code.

documentation in section 4.5 was on the project LOA (see table 4 in section 3.3), and while the documentation from section 4.2.2 was on the firm LOA, this is not the only source of information that can be looked at. Mainly, another source of information can be examined as well: ASF member[67] involvement in the projects.

Recall that ASF members are a group of committers and PMC members that annually decide on the composition of the Board of Directors. Thus, being either a committer or PMC member, they have the rights to make (permanent) changes to the code base of a project. In addition, these ASF members are affiliated with the ASF and list the ASF as the firm that they work for on GitHub[68]. While these ASF members work toward the paths that are in the best interest of the projects (recall the independence of the ASF projects under its current governance model), and no individuals affiliated with the ASF receive a standard monetary compensation (i.e., a salary) for their efforts[69], they do adhere to the Apache way of working[70] and the PMC chairs (and by extension the PMC members) are accountable to the Board of Directors. It thus might be insightful to examine the role these ASF members play in the everyday functioning of a project, as it could provide (indirect) insights into the influence of the Board of Directors (and by extension the ASF itself) on the projects. To do so, the presence (i.e., involvement) of these ASF members was examined for the most recent 100 merged pull requests per project, extracting information on these pull requests from GitHub with the help of the GitHub Graph-QL API[71]. The number of reviews done by ASF members was offset by the total number of reviews conducted per project, i.e., the total reviews conducted by all review participants. Recall that reviews are conducted in pull requests by committers and PMC members (in this case, only those that are ASF members) to test the form and applicability of the code that is submitted through a pull request. In addition, note that multiple reviews within a single pull request can be executed by the same ASF member or non-ASF member. The results of this analysis can be found in figure 6, where the gray bars represent the reviews conducted by ASF members, and the black bars represent the total number of reviews conducted.

The graph shows that the involvement of ASF members in pull requests (at least with respect to reviewing the code) varied widely across projects. Only the Airflow and Pulsar projects as well as the Kyuubi and Skywalking projects show some notion of similarity (the former having about 50% of its ASF members involved in the reviews and the latter around 80 to 90%). However, these projects all belong to different categories, as the Airflow project is in the incremental, unsustainable category, while the Pulsar and Kyuubi projects are in the incremental, sustainable category, and finally, the Skywalking project is in the radical, unsustainable category. Thus, no propositions can be formed based on this information.

4.7.2 Sponsorship

For the involvement of external firms in the project, one has to look at sponsorships. As stated before, the developers that work on the ASF projects do not receive monetary compensation for their efforts from the ASF, regardless of the role (i.e., contributor, committer, PMC member etc.) that they fulfil. It is thus not possible to find any direct sponsorship compensation from the ASF to its developers (as it does not exist). This was confirmed by the author when looking through the data on sponsorships found on the GitHub API. In this process, however, two interesting thoughts surfaced. For once, it could be interesting to determine how sponsorships are handled by other external firms.

[67] Note that collaborators are not examined here, as they are not directly affiliated to the ASF itself, but only have access to certain repositories (i.e., projects). Also see this page: https://docs.github.com/en/organizations/managing-user-access-to-your-organizations-repositories/adding-outside-collaborators-to-repositories-in-your-organization.

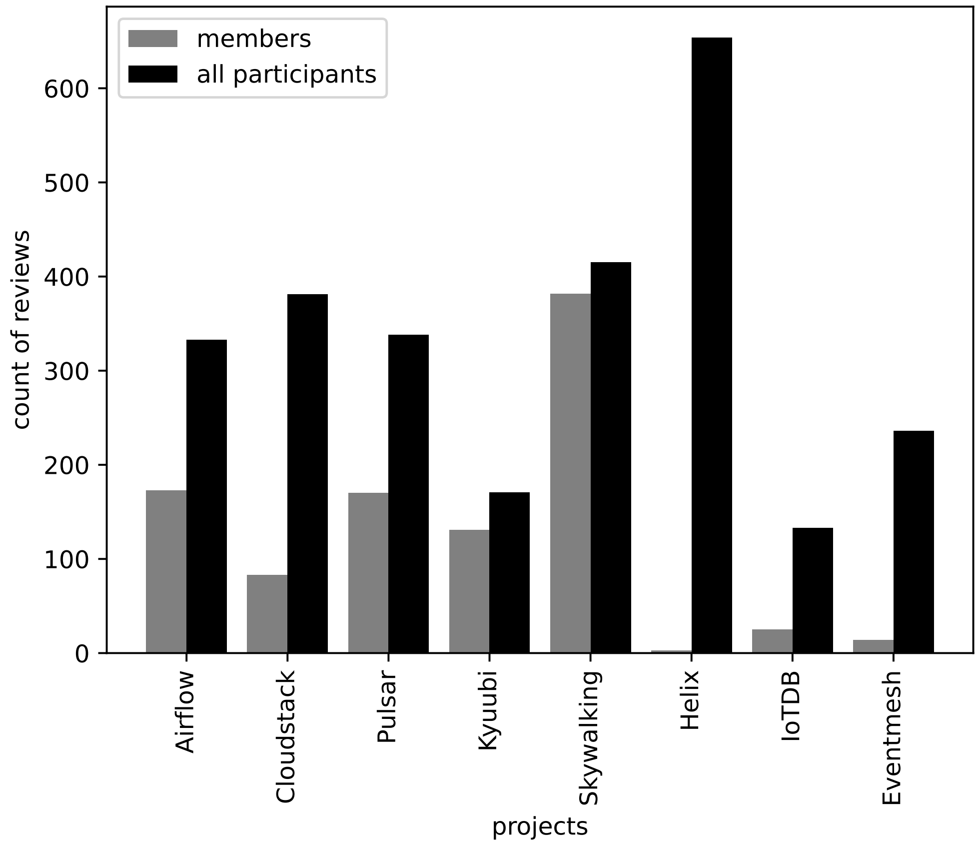
[68] See for example the profile of the following PMC member of the Airflow project: https://github.com/potiuk.

[69] https://www.apache.org/foundation/how-it-works.html

[70] https://www.apache.org/theapacheway/index.html

[71] Graph-QL is a query language for extracting data from APIs. It was developed as a response to the tedious process of executing multiple requests on the traditional REST API.

Figure 6: Reviews ASF Projects

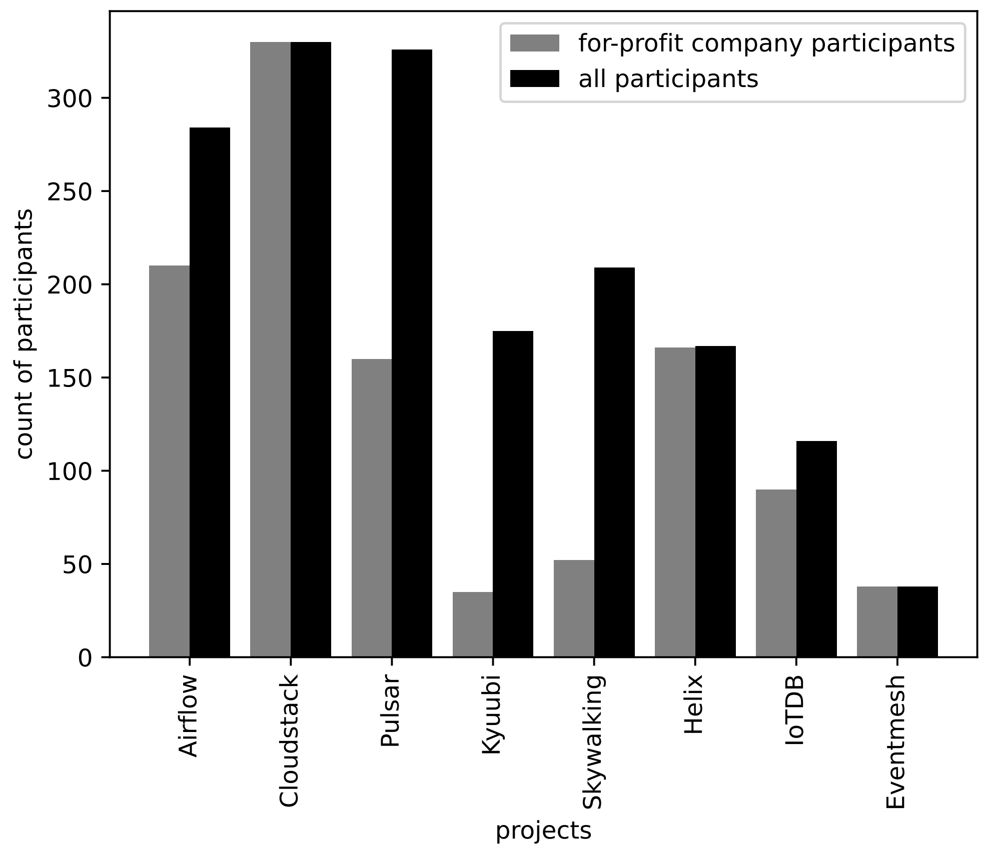


While the ASF does not pay its developers directly, an external firm (so, any for-profit firm that is not the ASF) does have this option. For example, it can occur that such an external firm uses the product of one of the ASF's projects (say a piece of software) and thus has an interest in maintaining and upgrading this software in the interest of the firm. One way to accomplish this maintenance and continued development is by paying internal employees (so, individuals that work for this for-profit firm) to work on the OSS project during work hours. To determine what the proportion of such individuals was on the projects, figures 7 and 8 were constructed.

These figures show the number of for-profit firm employees (i.e., external firm employees) present in the 100 most recent closed issues and merged pull requests per project, offset by the total number of participants on these issues and pull requests. To determine if the participants were from these external firms, the GitHub profiles of those participants were examined to see if they listed a for-profit firm as their place of employment, similar to what was done with the ASF members from section 4.7.1. Note that participants are only counted once per discussion in the figures, but of course, can occur in multiple discussions. While the author would have preferred to examine the reviews on pull requests for these individuals (similar to what was done in figure 6), the fields that the API uses did not allow such a selection. After some examination by the author, it became clear why: Individuals that are affiliated with an external firm (and thus work on the project on the firms behalf) can, at the same time, also be members of the ASF, see for example the GitHub profile of developer Xiaodong Deng[72]. Thus, while figure 6 remains valid (as it does not matter where an ASF member originates from, as long as he or she is an ASF member), figure 7 cannot be constructed like

[72] https://github.com/XD-DENG

Figure 7: Pull Request Participants ASF Projects



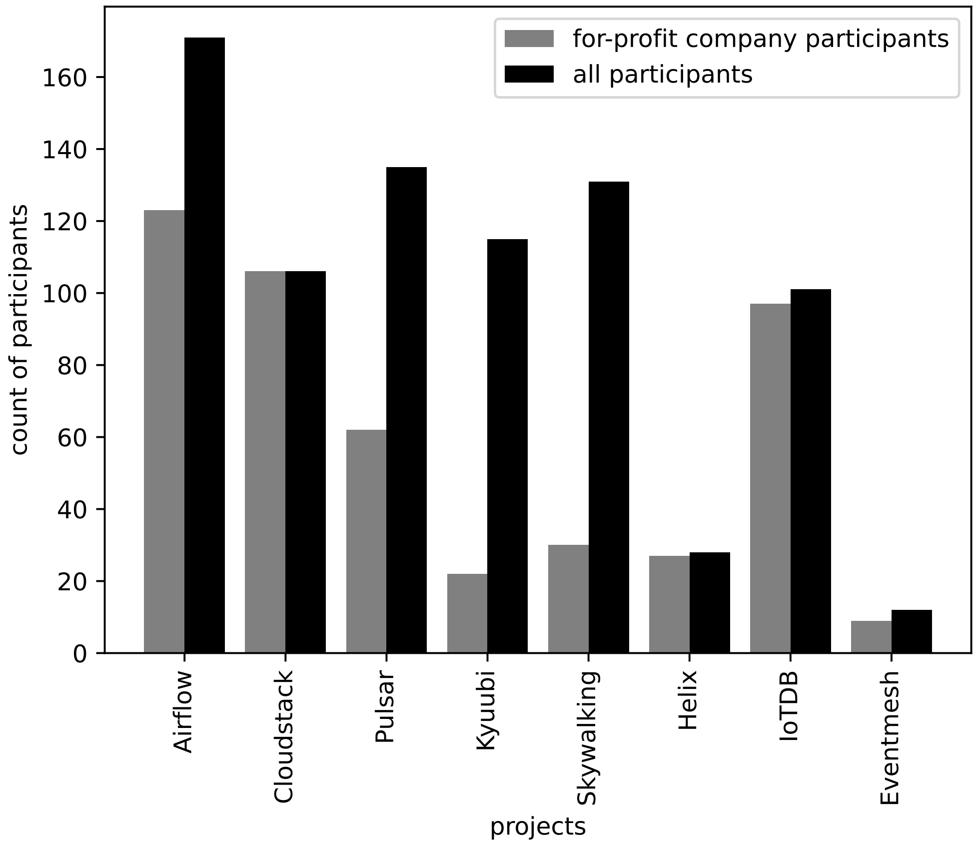
this, as there is no option within the API to determine if an individual works for another firm and is an ASF member. Instead, the graphs show the number of for-profit firm employees, regardless of being ASF members or not. As a final note before the graphs are discussed, it should be stated that the author cannot determine with certainty if the firms that the individuals list as their places of work directly compensate them by working on the projects as this information is not disclosed due to (presumably) privacy regulations. However, the author believes that it is safe to assume so, as these individuals that work for other firms often directly state the involvement of their employers in discussion threads on the platform. See the following comments for some examples, where *emphasis* is placed on terms that link this comment (and the one who posted it) to a firm:

> “I can confirm that one of *our customers* also faced a similar issue with poke=“reschedule” and increasing poke interval had fixed the issue for them.”

> “@turbaszek I am currently testing Airflow v2.0.0b3 against the same DAGS we currently *run on production* against 1.10.12 and I can confirm that this is still an issue.”

Focusing on the graphs (i.e., figure 7 and 8), it becomes clear that, like the ASF members examined in figure 6 in section 4.7.1, the involvement of for-profit employees in issue and pull request discussions differs greatly from project to project. However, unlike the results found in figure 6, some consistencies per category can be found when the graphs are placed side-to-side. For once, both graphs display cross-graph consistency; in other words, if many external firm employees participated in the discussions on pull requests for a particular project, they also did so for the discussions

Figure 8: Issue Participants ASF Projects



on issues, and vice versa. Second, within the categories of incremental, unsustainable (so, looking at the Airflow and Cloudstack projects) and radical, sustainable (so, looking at the IoTDB and Eventmesh projects), the participation of for-profit employees (i.e., external firm employees) in the discussions (both for issues and pull requests) was high or nearly all participants were for-profit employees. Thus, the following propositions can be made:

Proposition 8: Participation of external firms in incremental and unsustainable projects is high.

Proposition 9: Participation of external firms in radical and sustainable projects is high.

Due to how the examination of the pull requests was conducted, the author could not examine the nature of the participation (i.e., in which way are the external firm employees involved), but could only confirm that external firm employees participated.

Recall that two trains of thought occurred when examining the direct sponsorship of the ASF to its members. The first is addressed in the above graphs and subsequent propositions, but the second has yet to be discussed. It is about sponsorship within the ASF, but not the sponsorship of the ASF to its members. Instead, the author deemed it interesting to also look at the sponsorship that the ASF as a foundation receives from other firms and external parties. The ASF provides documentation on this on its website[73], which was examined here. First, as became clear from

[73]https://www.apache.org/foundation/sponsorship;
https://www.apache.org/foundation/thanks;
https://www.apache.org/foundation/contributing.html#CorporateGiving

this document as well, is that while the ASF does not directly sponsor its members, it does use the sponsorships that it receives to fund projects indirectly, for example by providing network infrastructure. It would be very interesting to see how sponsorship incomes are divided among projects based on certain characteristics, such as project size, “contribution to the ASF”, users, and ASF member counts, etc. However, this information is not disclosed by the ASF and as a result no further propositions can be made.

5 Theoretical Implications

This section functions as an overview of the attribute sections and their results (i.e., the propositions), in addition to discussing the overall theoretical implications of this study. First, table 11 functions as an update to table 4 showing the same LOA and attribute definition, but updates the references column with any additional scientific studies that were addressed in their respective sections. In addition, a 4th column is provided which briefly outlines how the attribute was analysed in its respective section. Second, table 12 outlines the propositions found in this study.

Table 11: Open-Source Software Attributes (Updated)

Level-Of-Analysis	**Attribute and Definition**	**References**	**Attribute Analysis**
Developer	Nature of communication – What is the level of formality employed in the project's communication.	Crowston et al. (2012) Medappa and Srivastava (2019) Peng et al. (2013) Peng (2009) Casalo et al. (2009) Wu et al. (2016) Wu et al. (2023) Bellantuono et al. (2012) Howison and Crowston (2014)	Analysing the formality of long, closed issue discussions with the help of the Chat-GPT AI.
	Work structure – What is the style of working within a project with regard to superposition and co-work.	Crowston et al. (2012) Medappa and Srivastava (2019) Peng et al. (2013) Peng (2009) Casalo et al. (2009) Wu et al. (2016) Wu et al. (2023) Bellantuono et al. (2012) Howison and Crowston (2014)	Looking at superposition and co-work work structures by analysing release histories of the projects.

Continued on next page

Table 11: Open-Source Software Attributes (Updated) — Continued

Level-Of-Analysis	**Attribute and Definition**	**References**	**Attribute Analysis**
Project	Governance and leadership – What is the governance and leadership structure of a project with regard to democracy, transparency and structure.	Midha and Palvia (2012) Crowston et al. (2012) Sagers (2004) Neufeld and Gu (2019) Aberdour (2007) O'Mahony and Bechky (2008) O'Mahony and Ferraro (2007)	Examining documentation on the governance and leadership of projects.
	License type – Which license is tied to a project and how does the choice of license influence the project's outcomes.	Midha and Palvia (2012) Crowston et al. (2012) Gezici et al. (2019) Amrollahi et al. (2014) Subramaniam et al. (2009) J. A. Colazo et al. (2005) Showole et al. (2011) Stewart, Ammeter, et al. (2006) Sen et al. (2011) Lindman et al. (2010) J. Colazo and Fang (2009) J. A. Colazo et al. (2005)	Discussing the implications of licensing choice in the field of OSS.
Firm	Firm involvement – What is the level of control and sponsorship imposed on projects by the firm that owns the projects and by external firms.	Crowston et al. (2012) Medappa and Srivastava (2019) Deodhar et al. (2010) Lamastra (2009) Capra et al. (2011)	Investigating (external) firm involvement and sponsorship in OSS projects using the GitHub API and ASF documentation.

Finally, table 13 provides an update to table 3. Akin to the model by Nolan and McFarlan (2005) outlined in section 2.1, it details the differences that exist between the quadrants based on the found propositions of table 12. The author is aware that the quadrants could be named to give context to their contents, similar to how Nolan and McFarlan named their quadrants "Factory mode" or "Support mode" (see figure 2 in section 2.1). However, after careful examination of the quadrants, the author concluded that while differences between the quadrants exist (which will be discussed below), these differences are not large, and the amount of content present in the quadrants is relatively small (when compared to the work of Nolan and McFarlan). Thus, by giving names to the quadrants based on their contents and comparatively small differences, the author is at risk of pushing the quadrants in certain directions without fully understanding the dynamics of those directions. Hence this will not be done. Similarly, Nolan and McFarlan provide

Table 12: Open-Source Software Propositions

Attribute	Proposition(s)
Nature of communication	**Proposition 1**: The nature of communication varies from formal to informal based on the specific developer, context, and project examined.
Work structure	**Proposition 2**: The work structure of projects transitions from co-work to superpositioned over time.
	Proposition 3: The work structure of incremental projects can be typified as superpositioned.
	Proposition 4: The work structure of radical projects can be typified as co-work.
Governance and leadership	**Proposition 5**: The governance and leadership of incremental and unsustainable projects are autocratic and employ a high structure.
	Proposition 6: The governance and leadership of incremental and sustainable projects employ low transparency.
	Proposition 7: The governance and leadership of radical and unsustainable projects employ low transparency and structure.
License type	-
Firm involvement	**Proposition 8**: Participation of external firms in incremental and unsustainable projects is high.
	Proposition 9: Participation of external firms in radical and sustainable projects is high.

two “orientations” for their quadrants based on their need for new IT titled “defensive” and “offensive” (which were also briefly discussed in section 2.1.1). In order to replicate these orientations the author considered a similar structure, labelling sustainable projects are immature and unsustainable projects as mature, which was based on the conclusions derived in section 4.4.2. However, in the end it was chosen to not do this, as the author viewed these structures (i.e., maturity and immaturity) as inherent aspects to the projects, rather than overarching “themes” like the orientations provided by Nolan and McFarlan. Thus, in the end it was chosen to leave table 13 as is.

With the tables of this section in place, Research Question 2 can be answered: Which attributes can be identified in Open-Source Software projects? In addition, the contents and theoretical implications of the Open-Source Software Strategic Grid can be discussed. While the quadrants of the grid were not named, the information present in each cell can still give direction to firms and individuals looking to invest in certain OSS projects. For example, if an individual looking to invest in a project (e.g., develop a piece of code) values high transparency and structure with regard to governance and leadership, radical and unsustainable projects might receive less weight in the decision process because their transparency and structure are low. Similarly, a firm looking to invest in a mature project that receives support from other firms and has an autocratic leadership structure (which is more akin to a traditional firm) might find itself interested in projects in the incremental and unsustainable category. These implications make clear that the grid can be used by practitioners to gravitate toward certain categories (i.e., quadrants) of projects and understand possible antecedents to success (i.e., understand the attributes). However, the author would like to note that the implications discussed above along with the grid do not provide a complete picture. Note that the dynamics of the attributes captured in the propositions should not be considered in isolation—No single examined attribute or proposition can provide an encompassing overview. This becomes especially apparent when examining the contents

Table 13: Open-Source Software Strategic Grid (Updated)

		Innovation	
		Incremental	*Radical*
Sustainability	*Sustainable*	Projects in this quadrant will employ different natures of communication and their work structure can be typified as superpositioned. In addition, their transparency (with regard to governance and leadership) is low.	Projects in this quadrant will employ different natures of communication and their work structure can be typified as co-work, which will transition to a superpositioned work structure over time. In addition, external firm involvement (in the form of participation) is high for these projects.
	Unsustainable	Projects in this quadrant will employ different natures of communication and their work structure can be typified as superpositioned. In addition, their structure (with regards to governance and leadership) is high and autocratic. Finally, external firm involvement (in the form of participation) is high for these projects.	Projects in this quadrant will employ different natures of communication and their work structure can be typified as co-work, which will transition to a superpositioned work structure over time. In addition, their transparency and structure (with regard to governance and leadership) are low.

of propositions 1 and 2 from table 12, which can be found in all four quadrants. While combining these elements (i.e., these attributes) to reach a more complete understanding of the situation and the projects at hand, it cannot tell the entire story. In other words, while the grid found in table 13 contributes to providing managers and individuals with a "tool" for their exploration of the OSS landscape and the subsequent choice to invest in a certain (category of) project, it should never be used as the sole reason for that choice. Rather, it provides a solid foundation on which managers and individuals can build their own understanding of projects and how those projects operate within the environment in which they reside. By understanding the placement of a project (or a group of projects) in the framework, and analysing or estimating the perceived changing placement over time (in other words, when projects might change quadrants), one can adapt and change his strategy accordingly, similar to the placement of firms in the model of Nolan and McFarlan.

6 Conclusion

The discussion on the findings for each individual attribute has already taken place in the respective section of that attribute to fit the iterative way in which this study conducted research. In addition, the overall implications of this study are discussed in section 5, completing phases 1 and 2. As such, this section will be kept short and concise, providing a brief overview of the conducted study and mentioning some limitations and directions for further research. As the complexity of OSS projects evolves and the number of projects continues to grow, the investment choices that companies and individuals make become increasingly difficult. By iteratively examining a curated set of OSS projects through their attributes, some preliminary propositions about the functioning of projects within their respective categories (e.g., incremental, sustainable) were derived. These results provide managers and individuals with a "tool" that can aid their understanding of this complex landscape.

Starting the discussion on this study's limitations and directions for further research, the author has to acknowledge a limitation of the exploratory nature of this study. While care was put in the development and validation of the grid in section 2 (i.e., phase 1) and the development of the propositions in section 4.3 to section 4.7 (i.e., phase 2), no empirical validation is conducted on the resulting theory (i.e., the propositions), since this could not fit in this study's scope. In addition, no quantitative analysis of adequate proportion could be conducted to see if the perceived effects of the attributes on the projects held true when compared to all projects within the categories (see table 6). Further work needs to be conducted to close these gaps. This also highlights the time-restricted nature of this study (see for example the absence of constructs and propositions in section 4.6). The author would have preferred to have found more propositions, yet the data speaks for itself. More in-depth analysis of the attributes is required to increase the number of propositions found, which can further substantiate and refine the Open-Source Software Strategic Grid from section 5.

First, for the nature of communication, it would have been interesting to prompt the AI to make a ranking of all the examined issue discussions, ranking them from most to least formal. In this way, making propositions for the different quadrants might be feasible, for example if the two most formal projects (as rated by the AI's ranking) were in the same quadrant. Unfortunately, the unpaid version of the AI is unable to combine the information of different chats and for reasons specified in section 4.3.3, this information could not be combined into a single chat. Perhaps with the paid API that is tied to Chat-GPT or another language learning model (like IBM's Watson) this could have been done. Additionally, it would be interesting to investigate the tone and emotion of the issue discussions to determine if any differences per project could be found there. Second, to further test proposition 2 (which covered the work structure shift over time), one would want to select and examine projects that have a similar age for all four quadrants to see if the proposition holds and if other propositions could have been found. This was not done due to time constraints and because the choice to analyse certain projects was made at an earlier stage in phase 2. Third, for governance and leadership, the constructs of democracy, transparency, and structure could be further examined by holding interviews with the parties involved in these concepts (similar to what was done in the studies of O'Mahony and Bechky (2008) and O'Mahony and Ferraro (2007)). Fourth, by examining projects that have different licenses attached to them, one might be able to derive propositions for different quadrants. Finally, it was shown that firm involvement was present in certain quadrants in the form of participation, but the nature of that participation (i.e., how are the external firm employees involved, what do they do) could be examined as well.

Finally, some caveats with the development of the framework in section 2 need to be discussed. In section 2.2.1, no dichotomous categorisations could be found in the construction of the framework, lessening the support from the model by Nolan and McFarlan (2005) that it was derived from. In addition, recall that in the same section, a specific approach was taken to determine if a project was sustainable or not (in the context of this study)—If a project's Growth Rate is above the average for all projects and a project's Developer Score is above the average for all projects, then a project is considered sustainable, otherwise it is not. The reader should be aware that there are some caveats when taking this approach. The interpretability of the individual components that make up the scores of these dimensions (like the Core Developer Ratio) go down, as they are mixed to make up the Developer Score and the Growth Rate, which is further diluted by the combination of the latter. In addition, by combining the Developer Score and the Growth Rate, the operationalisation of the y-axis becomes nonlinear. This is also the case for the Innovation Score, albeit in a less severe matter. Another caveat presents itself in the form of discriminant validity, as one could argue that a project that has many core developers (i.e., those developers that are very involved in the development process and contribute the bulk of the code, often in the form of new or enhanced functionality) automatically has a higher innovation score (swaying to the radical innovation quadrants), as the contributions of core developers are often typified by a higher number of commits and file changes (at least in theory). In other words, the constructs are not completely empirically distinct, see also the work of Rönkkö and Cho (2022). However, after an extensive discussion between this study's research participants, it was decided that these components (i.e., the missing dichotomous categorisation, the individual interpretability of the components and their discriminant validity) could suffer in favour of the practicality that the approach provides as well as its strong translatability to the OSS context.

A Additional Tables

Table 14: Growth Rate Numerical Example

Year	Active Developers in Current Year	Growth Compared to the Year Before	Growth Rate
1	10		
2	15	(15/10) = 1.5	
3	17	(17/15) = 1.133	
4	24	(24/17) = 1.412	
Total			(1.5+1.133+1.412) / 3 = 1.348

Table 15: Open-Source Software Attributes (Reference)

Level-Of-Analysis	Attribute and Definition	References
Artifact	Software quality – What is the quality of the code.	S.-Y. T. Lee et al. (2009) Ghapanchi et al. (2011) Garomssa et al. (2022) Fernandes (2013) Haider et al. (2020)
	Software modularity – How modular is the code.	Midha and Palvia (2012) Crowston et al. (2012) Aberdour (2007)
	Software requirements analysis and planning – What techniques are used to plan out the next feature implementations for the code, if any.	Crowston et al. (2012)
	Software testing – How easy is it to test the code and how often is this done.	Crowston et al. (2012) Hanoğlu and Tarhan (2019) Aberdour (2007)
	Software maintenance and sustainability – How easy is it to maintain the code (also for the long run) and how often is this done.	Crowston et al. (2012) Ghapanchi (2015) Gezici et al. (2019)
	Software releases and project activity – How often is a new version released and how active is the development of the code.	Ghapanchi et al. (2011) Fernandes (2013) Crowston et al. (2012) Ghapanchi (2015) Amrollahi et al. (2014) Istiyanto et al. (2009)

Continued on next page

Table 15: Open-Source Software Attributes (Reference) — Continued

Level-Of-Analysis	**Attribute and Definition**	**References**
	Software re-use in other projects – How often does the code of one project get re-used for the code in another project.	Fernandes (2013)
	Software availability – Is the source code available or not.	Haider et al. (2020)
	Software development cost – What are the costs associated with developing the code.	Haider et al., 2020
	Technology use – What technologies and programming languages are used in the development of the code.	Crowston et al. (2012) Amrollahi et al. (2014) Daniel et al. (2018) Zanella and Liu (2020) Peng (2009)
Individual	Developer size – What is the number of developers on a specific project.	Midha and Palvia (2012) Crowston et al. (2012) Gezici et al. (2019) Amrollahi et al. (2014) Sen et al. (2012) Beecher et al. (2008)
	Developer growth – What is the growth in the number of developers on a specific project.	Midha and Palvia (2012) Istiyanto et al. (2009)
	Developer geographic location – What are the geographic locations of the developers.	Crowston et al. (2012)
	Developer motivation – What are the motivations of the developers to participate on a specific project.	Crowston et al. (2012) Medappa and Srivastava (2019) Fernandes (2013)
	Developer knowledge – What knowledge do the developers bring to the table.	Peng (2009)
	Developer interest – How interested are the developers in certain projects.	Ghapanchi (2015) Subramaniam et al. (2009)

Continued on next page

Table 15: Open-Source Software Attributes (Reference) — Continued

Level-Of-Analysis	**Attribute and Definition**	**References**
	Developer coordination and collaboration – How do developers communicate and collaborate within and between projects.	Crowston et al. (2012) Medappa and Srivastava (2019) Peng et al. (2013) Peng (2009) Casalo et al. (2009) Wu et al. (2016) Wu et al. (2023) Bellantuono et al. (2012)
	Developer core vs. peripheral – What is the ratio of core versus peripheral developers.	Daniel et al. (2018) Crowston and Shamshurin (2017) Aberdour (2007)
	Developer ability to fix bugs – How effective and efficient are developers in fixing code bugs.	Haider et al. (2020) Gezici et al. (2019) Amrollahi et al. (2014)
	Developer language differences – How do the languages spoken and used by developers differ.	Daniel et al. (2018)
	Developer recognition by peers – What is the level of recognition that developers receive by peers.	Fernandes (2013)
	Developer awareness of other projects – How aware are developers about the existence of projects (other than the ones they are working on).	Daniel and Stewart (2017)
	Developer role differences – How do the different roles of developers differ.	Daniel et al. (2018)
Group	Language translation - Translation of the software to native languages.	Midha and Palvia (2012)
	Governance and leadership – How is the project ran, who makes the shots and who can contribute and bring up new ideas.	Midha and Palvia (2012) Crowston et al. (2012) Sagers (2004) Neufeld and Gu (2019) Aberdour (2007)

Continued on next page

Table 15: Open-Source Software Attributes (Reference) — Continued

Level-Of-Analysis	Attribute and Definition	References
	License type – Which license is tied to the project and how does it affect the way in which the project develops.	Midha and Palvia (2012) Crowston et al. (2012) Gezici et al. (2019) Amrollahi et al. (2014) Subramaniam et al. (2009) J. A. Colazo et al. (2005) Showole et al. (2011)
	Security investment early development – How much is invested in the security of a project in the initial stages of its development.	Chehrazi et al. (2016)
	Successful project initiation – How successful is the initiation of a project in terms of survivability.	Showole et al. (2011) L. Wang and Wang (2020)
	Project ranking – How does a project rank (in terms of popularity or adoption) when compared to other projects.	Istiyanto et al. (2009)
	Project age – How mature is the project.	Istiyanto et al. (2009)
	Intellectual property rights – What are the IPR's tied to a project.	Wen et al. (2013)
Organisation	Firm involvement – How is the firm involved in the development and governance of the project.	Crowston et al. (2012) Medappa and Srivastava (2019) Deodhar et al. (2010) Lamastra (2009)
	Project connectedness – In what way and how much are projects connected to one another.	Daniel et al. (2018)
Societal	User interest – What is the interest shown by the user community.	Ghapanchi et al. (2011) Ghapanchi (2015) Subramaniam et al. (2009)
	User satisfaction – How satisfied are the users in the community with the product.	S.-Y. T. Lee et al. (2009) Gezici et al. (2019) Garomssa et al. (2021)

Continued on next page

Table 15: Open-Source Software Attributes (Reference) — Continued

Level-Of-Analysis	**Attribute and Definition**	**References**
	User net benefits – Which net benefits do the users in the community derive from the product (net benefits are seen here as job- and decision-making performance).	S.-Y. T. Lee et al. (2009)
	User service support – What is the level of community service quality.	S.-Y. T. Lee et al. (2009)
	User adoption – What is the number of users present in the community that adopted (i.e., use) the product.	S.-Y. T. Lee et al. (2009) Midha and Palvia (2012) Gezici et al. (2019) Amrollahi et al. (2014) Sen et al. (2012) Beecher et al. (2008) Fernandes (2013) Foushee et al. (2013) Garomssa et al. (2021) Fitzgerald (2009)

Table 16: Formality Nature of Communication

Project	**Issue**	**Chat-GPT Response**
Airflow	#19891	Based on the assessment of formality for the provided text prompts, it appears that the overall weighted notion of formality leans towards the informal side. There were several instances of informal language, abbreviations, casual expressions, and the use of emojis. However, it's important to note that formality can vary depending on the context, audience, and communication platform.
	#13542	Based on the prompts and responses in this chat, the weighted notion of formality leans towards an informal tone. While some responses exhibit a more formal language style, the majority of the interactions tend to be casual and conversational.

Continued on next page

Table 16: Formality Nature of Communication — Continued

Project	Issue	Chat-GPT Response
	#18041	Based on the evaluation of the various text prompts in this chat, the overall weighted notion of formality is moderately informal. While some messages exhibited a more formal tone, the majority of the texts leaned towards informality. This assessment is based on factors such as the use of abbreviations, casual language, lack of proper greetings and closures, sentence fragments, and the occasional absence of punctuation and capitalization. While formality can vary depending on the context and the audience, the majority of the texts in this conversation were informally phrased.
Cloudstack	#3179	Based on the responses provided throughout this conversation, the overall weighted notion of formality is moderately formal. The messages mostly contain professional language, proper grammar and punctuation, and appropriate use of technical terms. However, there were occasional instances of informal language and casual expressions. Overall, the tone maintained a professional and respectful demeanor.
	#5204	Based on the analysis of the responses provided in this chat, the overall weighted notion of formality is moderately informal. While some responses exhibit a more formal tone, a significant number of responses lean towards informality in terms of language and style. It is important to note that formality can vary depending on the context and audience, and the rating provided here is based on a general assessment.
	#6836	Based on the prompts and their corresponding ratings, the overall notion of formality in the text samples leans towards informal to moderately formal. The usage of informal language, casual tone, and abbreviations is observed in some texts, while others demonstrate a more formal tone with proper structure and professional language. However, it is important to note that formality can be subjective and context-dependent, and different individuals may perceive formality differently.

Continued on next page

Table 16: Formality Nature of Communication — Continued

Project	Issue	Chat-GPT Response
Pulsar	#6054	Based on the responses to the prompts in this chat, the overall weighted notion of formality in the provided statements leans towards the informal side.
	#17097	Based on the assessment of formality in the provided prompts and responses, the overall weighted notion of formality in this chat leans towards the informal side. While there were a few instances of more formal language, the majority of the text exhibited an informal tone, with casual expressions, simple sentence structures, and a conversational style. In more formal or professional contexts, it would be advisable to use a more formal tone and structure to ensure appropriate communication.
	#180	Based on the provided prompts and their respective formality ratings, the overall weighted notion of formality in this conversation tends to lean towards informality. While some prompts received higher formality ratings, the majority of the prompts were rated as informal or moderately informal. This indicates that the conversation as a whole exhibits a more casual and relaxed tone. However, it's important to note that formality can vary depending on the context, audience, and specific language choices used within a conversation.
Kyuubi	#838	Based on the responses to the prompts in this chat, the overall weighted notion of formality leans towards a moderately formal tone. While some responses were more informal, particularly in shorter and casual sentences, the majority of the responses demonstrated a moderate level of formality. The use of proper grammar, terminology, and elaboration in most cases contributed to the formal tone. However, it's important to note that the formality of language can vary depending on the context, and individual perceptions of formality may differ.

Continued on next page

Table 16: Formality Nature of Communication — Continued

Project	**Issue**	**Chat-GPT Response**
	#32	Based on the analysis of the provided text throughout this conversation, the overall weighted notion of formality tends to lean towards a moderately informal style. While there are occasional formal elements, such as discussing technical concepts and using proper grammar, the frequent use of abbreviations, informal language, and conversational tone indicates a more relaxed and casual approach to communication. It is important to note that the formality can vary depending on the specific prompts and context of the conversation.
	#304	Based on the responses provided throughout the conversation, the overall weighted notion of formality in the text leans towards being moderately informal. While there are instances where the language used is relatively formal, there are also numerous instances where casual language, abbreviations, and informal expressions are present. It is important to note that formality can vary depending on the context, audience, and purpose of communication.
Skywalking	#9556	Based on the responses to the prompts in this chat, the weighted notion of formality is primarily informal. Many of the provided texts contained casual language, abbreviations, and informal expressions. While there were some instances of more formal language, the overall trend leaned towards informality. It's important to note that formality can vary depending on the context, and certain informal language may be appropriate in casual or informal conversations.
	#9376	Based on the responses given to various prompts throughout the conversation, the overall weighted notion of formality is moderately informal. While there were instances of more formal language, the majority of the responses leaned towards casual and informal communication.

Continued on next page

Table 16: Formality Nature of Communication — Continued

Project	**Issue**	**Chat-GPT Response**
	#1666	Based on the ratings given to the various prompts in this chat, the overall weighted notion of formality tends to be informal. The majority of the text samples exhibited informal language, casual expressions, and a lack of technical precision. While some messages had a more formal tone, they were outnumbered by the informal ones. It is important to note that the formality of the responses can be influenced by the nature of the conversation and the context in which the language is used.
Helix	#609	Based on the responses provided in this chat, it can be inferred that the notion of formality has been given a moderate weight. While some responses aimed to increase formality by suggesting alternative phrasings and expressions, the overall trend leaned towards a more informal tone in the text. This may reflect a balanced approach where there is a recognition of the need for formality in certain contexts, but also an understanding that informality can be acceptable in certain situations or among peers.
	#349	Based on the answers provided throughout this conversation, the weighted notion of formality leans towards moderate informality. While some statements were rated as moderately formal, the majority of the text exhibited informal characteristics, such as casual language, conversational tone, and the absence of explicit formality markers. It is important to note that formality can vary depending on the context, and the assessment of formality is subjective to some extent.

Continued on next page

Table 16: Formality Nature of Communication — Continued

Project	Issue	Chat-GPT Response
	#1336	Based on the answers provided in this chat, the weighted notion of formality leans towards a moderate level. While some responses indicated a higher level of formality, particularly in technical or professional contexts, the majority of the responses reflected a moderate level of formality. The use of technical terminology, direct language, and structured explanations contributed to a sense of formality. However, occasional informal expressions, casual tones, and the absence of highly formal elements balanced out the overall formality level. Thus, the weighted notion of formality in this chat tends to be moderate.
IoTDB	#2124	Based on the responses provided in this chat, the overall weighted notion of formality tends to lean towards the informal side. While some responses exhibit a moderate level of formality, many responses use casual language, contractions, and informal expressions. To enhance formality, it is advisable to use complete sentences, avoid casual language and contractions, and employ more formal greetings and expressions where appropriate.
	#2217	Based on the responses to the prompts in this chat, the weighted notion of formality can be described as moderately formal. The majority of the responses fell within this range, with a mix of formal and informal elements. The use of polite language, technical terminology, and complete sentences contributed to the formality, while occasional casual greetings and conversational tone added informality to the overall communication style.
	#9326	Based on the responses provided, the weighted notion of formality in this chat leans towards being relatively informal. While some texts exhibit a more formal tone, the majority of the responses indicate a casual and conversational style. It's important to note that the formality of language can vary based on context, and different communication styles may be appropriate in different situations.

Continued on next page

Table 16: Formality Nature of Communication — Continued

Project	**Issue**	**Chat-GPT Response**
Eventmesh	#274	Based on the responses to the prompts in this chat, the weighted notion of formality leans towards the informal side. The majority of the texts included casual language patterns, informal greetings, and the use of "@" symbols to mention specific individuals. While some texts contained technical terms, they were often accompanied by informal language or sentence structures. Overall, the conversations maintained a relatively casual tone throughout.
	#435	Based on the responses to the prompts in this chat, the weighted notion of formality leans towards informality. The majority of the evaluated texts were rated as informal, displaying characteristics such as the use of casual language, contractions, emoticons, and conversational tone. However, it is important to note that the formality of a conversation can vary depending on the context, participants, and specific communication platform being used.
	#349	Based on the evaluation of the provided text throughout this conversation, the overall weighted notion of formality can be considered moderate. While there were instances of relatively low formality, such as the use of "@" symbol for mentions and casual language, the majority of the text demonstrated a professional tone and used technical terminology appropriately. The formality level could be further improved by addressing issues like grammar, punctuation, and providing clearer and more structured explanations.

References

Aberdour, M. (2007). Achieving Quality in Open-Source Software. *IEEE software*, *24*(1), 58–64. https://doi.org/10.1109/MS.2007.2

Adewumi, A., Misra, S., Omoregbe, N., & Sanz, L. F. (2019). Fosses: Framework for open-source software evaluation and selection. *Software: Practice and Experience*, *49*(5), 780–812. https://doi.org/10.1002/spe.2682

Aksulu, A., & Wade, M. R. (2010). A Comprehensive Review and Synthesis of Open Source Research. *Journal of the Association for Information Systems*, *11*(11), 576–656.

Alenezi, M., & Almustafa, K. (2015). Empirical Analysis of the Complexity Evolution in Open-Source Software Systems. *International Journal of Hybrid Information Technology*, *8*(2), 257–266. https://doi.org/10.14257/ijhit.2015.8.2.24

Alenezi, M., & Khellah, F. (2015). Evolution Impact on Architecture Stability in Open-Source Projects. *International Journal of Cloud Applications and Computing (IJCAC)*, *5*(4), 24–35. https://doi.org/10.4018/IJCAC.2015100102

Amrollahi, A., Khansari, M., & Manian, A. (2014). How Open Source Software Succeeds? A Review of Research on Success of Open Source Software. *6*(2), 67–77.

Andriole, S. J. (2009). Boards of Directors and Technology Governance: The Surprising State of the Practice. *Communications of the Association for Information Systems*, *24*(1), 373–394. https://doi.org/10.17705/1CAIS.02422

Anwar, N., & Masrek, M. N. (2013). The Impact of IT Infrastructure Flexibility on Strategic Utilization on Information Systems: A Conceptual Framework. *2013 International Conference on Advanced Computer Science Applications and Technologies*, 510–514. https://doi.org/10.1109/ACSAT.2013.106

Banks, D., Leonard, C., Narayan, S., Thompson, N., Kramer, B., & Korkmaz, G. (2022). Measuring the Impact of Open Source Software Innovation Using Network Analysis on Github Hosted Python Packages. *2022 Systems and Information Engineering Design Symposium (SIEDS)*, 110–115. https://doi.org/10.1109/SIEDS55548.2022.9799290

Bart, C., & Turel, O. (2010). IT and the Board of Directors: An Empirical Investigation into the "Governance Questions" Canadian Board Members Ask about IT. *Journal of Information Systems*, *24*(2), 147–172. https://doi.org/10.2308/jis.2010.24.2.147

Beecher, K., Boldyreff, C., Capiluppi, A., & Rank, S. (2008). Evolutionary Success of Open Source Software: An Investigation into Exogenous Drivers. *Electronic Communications of the EASST*, *8*, 1–14.

Behfar, S. K., Turkina, E., & Burger-Helmchen, T. (2017). Network tie structure causing OSS group innovation and growth. *Problems and Perspectives in Management*, *15*(1), 7–18. https://doi.org/10.21511/ppm.15(1).2017.01

Bellantuono, N., Pontrandolfo, P., Scozzi, B., et al. (2012). Proximity, Innovation, and Success within Free/Libre Open Source Software development projects. In *Innovation, technology and knowledge* (pp. 125–156). Routledge Taylor & Francis Group.

Berente, N., Seidel, S., & Safadi, H. (2019). Research Commentary—Data-Driven Computationally Intensive Theory Development. *Information Systems Research*, *30*(1), 50–64. https://doi.org/10.1287/isre.2018.0774

Blok, V. (2021). What Is Innovation? Laying the Ground for a Philosophy of Innovation. *Techné: Research in Philosophy and Technology*, *25*(1), 72–96. https://doi.org/10.5840/techne2020109129

Booth, A., & Carroll, C. (2015). How to build up the actionable knowledge base: The role of 'best fit'framework synthesis for studies of improvement in healthcare. *BMJ quality & safety*, *24*(11), 700–708. https://doi.org/10.1136/bmjqs-2014-003642

Borges, H., Hora, A., & Valente, M. T. (2016). Predicting the Popularity of GitHub Repositories. *Proceedings of the The 12th international conference on predictive models and data analytics in software engineering*, (9), 1–10. https://doi.org/10.1145/2972958.2972966

Borges, H., & Valente, M. T. (2018). What's in a GitHub Star? Understanding Repository Starring Practices in a Social Coding Platform. *Journal of Systems and Software*, *146*, 112–129. https://doi.org/10.1016/j.jss.2018.09.016

Bouktif, S., Sahraoui, H., & Ahmed, F. (2014). Predicting Stability of Open-Source Software Systems Using Combination of Bayesian Classifiers. *ACM Transactions on Management Information Systems (TMIS)*, *5*(1), 1–26. https://doi.org/10.1145/2555596

Briest, G., Lukas, E., Mölls, S. H., & Willershausen, T. (2020). Innovation speed under uncertainty and competition. *Managerial and Decision Economics*, *41*(8), 1517–1527. https://doi.org/10.1002/mde.3199

Cabot, J., Izquierdo, J. L. C., Cosentino, V., & Rolandi, B. (2015). Exploring the Use of Labels to Categorize Issues in Open-Source Software Projects. *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 550–554. https://doi.org/10.1109/SANER.2015.7081875

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

Casalo, L. V., Cisneros, J., Flavián, C., & Guinaliu, M. (2009). Determinants of success in open source software networks. *Industrial Management & Data Systems*, *109*(4), 532–549. https://doi.org/10.1108/02635570910948650

Chehrazi, G., Heimbach, I., & Hinz, O. (2016). The impact of security by design on the success of open source software. *Twenty-Fourth European Conference on Information Systems (ECIS)*.

Chengalur-Smith, I., Sidorova, A., & Daniel, S. L. (2010). Sustainability of Free/Libre Open Source Projects: A Longitudinal Study. *Journal of the Association for Information Systems*, *11*(11), 657–683. https://doi.org/10.17705/1jais.00244

Colazo, J., & Fang, Y. (2009). Impact of License Choice on Open Source Software Development Activity. *Journal of the American Society for Information Science and Technology*, *60*(5), 997–1011. https://doi.org/10.1002/asi.21039

Colazo, J. A., Fang, Y., & Neufeld, D. (2005). Development Success in Open Source Software Projects: Exploring the Impact of Copylefted Licenses. *Americas Conference on Information Systems (AMCIS)*.

Colombo, M. G., Piva, E., & Rossi-Lamastra, C. (2014). Open innovation and within-industry diversification in small and medium enterprises: The case of open source software firms. *Research policy*, *43*(5), 891–902. https://doi.org/10.1016/j.respol.2013.08.015

Conlon, M. P. (2007). An examination of initiation, organization, participation, leadership, and control of successful open source software development projects. *Information Systems Education Journal*, *5*(38), 1–13.

Crowston, K., & Shamshurin, I. (2017). Core-periphery communication and the success of free/libre open source software projects. *Journal of Internet Services and Applications*, *8*, 1–11. https://doi.org/10.1186/s13174-017-0061-4

Crowston, K., Wei, K., Howison, J., & Wiggins, A. (2012). Free/Libre Open-Source Software Development: What We Know and What We Do Not Know. *ACM Computing Surveys (CSUR)*, *44*(2), 1–35. https://doi.org/10.1145/2089125.2089127

Daniel, S., Midha, V., Bhattacherhjee, A., & Singh, S. P. (2018). Sourcing knowledge in open source software projects: The impacts of internal and external social capital on project success. *The Journal of Strategic Information Systems*, *27*(3), 237–256. https://doi.org/10.1016/j.jsis.2018.04.002

Daniel, S., & Stewart, K. (2017). Implications of Alter Project Resources and Participant Roles for Open Source Software Project Commercial Success. *International Conference on Information Systems (ICIS)*.

Darcy, D. P., Daniel, S. L., & Stewart, K. J. (2010). Exploring Complexity in Open Source Software: Evolutionary Patterns, Antecedents, and Outcomes. *2010 43rd Hawaii International Conference on System Sciences*, 1–11.

De Noni, I., Ganzaroli, A., & Orsi, L. (2011). The Governance of Open Source Software Communities: An Exploratory Analysis. *Journal of Law and Governance*, *6*(1), 1–18. https://doi.org/10.15209/jbsge.v6i1.195

Deodhar, S. J., Saxena, K., & Ruohonen, M. (2010). Firm-Oriented Success Factors of an Open Source Software (OSS) Product. *Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, 1–4. https://doi.org/10.1145/1833272.1833273

Di Gangi, P., Teigland, R., & Yetis, Z. (2022). How do different stakeholder groups within an open source software project influence the project's development: A case study of OpenSimulator. *Information Technology & People*, (ahead-of-print). https://doi.org/10.1108/ITP-10-2021-0751

Dibrell, C., Davis, P. S., & Craig, J. (2008). Fueling Innovation through Information Technology in SMEs. *Journal of Small Business Management*, *46*(2), 203–218. https://doi.org/10.1111/j.1540-627X.2008.00240.x

Dong, J. Q., Wu, W., & Zhang, Y. S. (2019). The faster the better? Innovation speed and user interest in open source software. *Information & Management*, *56*(5), 669–680. https://doi.org/10.1016/j.im.2018.11.002

Drechsler, A., & Weißschädel, S. (2018). An IT strategy development framework for small and medium enterprises. *Information Systems and e-Business Management*, *16*, 93–124. https://doi.org/10.1007/s10257-017-0342-2

Drnevich, P. L., & Croson, D. C. (2013). Information Technology and Business-Level Strategy: Toward an Integrated Theoretical Perspective. *MIS Quarterly*, *37*(2), 483–509.

Ebert, C. (2009). Guest Editor's Introduction: How Open Source Tools Can Benefit Industry. *IEEE software*, *26*(2), 50–51. https://doi.org/10.1109/MS.2009.38

Eckert, R., & Mueller, A. (2017). Sustainability and Diversity of Open Source Software Communities. *International Conference on Software Engineering Advances (ICSEA)*, 48–54.

Fernandes, S. S. (2013). Quality, Success, Communication and Contribution in Open Source Software. *Electronic Communications of the EASST*, *48*. https://doi.org/10.14279/tuj.eceasst.48.816

Fitzgerald, B. (2009). Open Source Software Adoption: Anatomy of Success and Failure. *International Journal of Open Source Software and Processes (IJOSSP)*, *1*(1), 1–23.

Foege, J. N., Lauritzen, G. D., Tietze, F., & Salge, T. O. (2019). Reconceptualizing the paradox of openness: How solvers navigate sharing-protecting tensions in crowdsourcing. *Research Policy*, *48*(6), 1323–1339. https://doi.org/10.1016/j.respol.2019.01.013

Foushee, B., Krein, J. L., Wu, J., Buck, R., Knutson, C. D., Pratt, L. J., & MacLean, A. C. (2013). Reflexivity, Raymond, and the Success of Open Source Software Development: A SourceForge Empirical Study. *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, 246–251. https://doi.org/10.1145/2460999.2461036

Gamalielsson, J., & Lundell, B. (2014). Sustainability of Open Source software communities beyond a fork: How and why has the LibreOffice project evolved? *Journal of Systems and Software*, *89*, 128–145. https://doi.org/10.1016/j.jss.2013.11.1077

Garcia, R., & Calantone, R. (2002). A critical look at technological innovation typology and innovativeness terminology: A literature review. *Journal of Product Innovation Management*, *19*(2), 110–132. https://doi.org/10.1111/1540-5885.1920110

Garomssa, S. D., Kannan, R., Chai, I., & Riehle, D. (2022). How Software Quality Mediates the Impact of Intellectual Capital on Commercial Open-Source Software Company Success. *IEEE Access*, *10*, 46490–46503. https://doi.org/10.1109/ACCESS.2022.3170058

Garomssa, S. D., Kannan, R., & Chai, I. (2021). Updated DeLone and McLean IS Success Model and Commercial Open Source Software (COSS) Company Success. *Proceedings of the 10th Knowledge Management International Conference (KMICe2021), Putrajaya, Malaysia*, 33–38.

Gatignon, H., Tushman, M. L., Smith, W., & Anderson, P. (2002). A Structural Approach to Assessing Innovation: Construct Development of Innovation Locus, Type, and Characteristics. *Management science*, *48*(9), 1103–1122. https://doi.org/10.1287/mnsc.48.9.1103.174

Geiger, R. S., Howard, D., & Irani, L. (2021). The Labor of Maintaining and Scaling Free and Open-source Software Projects. *Proceedings of the ACM on Human-Computer Interaction*, *5*(CSCW1), 1–28. https://doi.org/10.1145/3449249

Gezici, B., Özdemir, N., Yılmaz, N., Coşkun, E., Tarhan, A., & Chouseinoglou, O. (2019). Quality and Success in Open Source Software: A Systematic Mapping. *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 363–370. https://doi.org/10.1109/SEAA.2019.00062

Ghapanchi, A. H. (2015). Investigating the Interrelationships among Success Measures of Open Source Software Projects. *Journal of Organizational Computing and Electronic Commerce*, *25*(1), 28–46. https://doi.org/10.1080/10919392.2015.990775

Ghapanchi, A. H., Aurum, A., & Low, G. (2011). A taxonomy for measuring the success of open source software projects. *First Monday*, *16*(8), 1–12. https://doi.org/10.5210/fm.v16i8.3558

Greenhalgh, T., Robert, G., Macfarlane, F., Bate, P., & Kyriakidou, O. (2004). Diffusion of Innovations in Service Organizations: Systematic Review and Recommendations. *The Milbank Quarterly*, *82*(4), 581–629. https://doi.org/10.1111/j.0887-378X.2004.00325.x

Haider, S., Khalil, W., Rashid, N., & Khalid, S. (2020). Success Factors for Open-Source Software Development from Vendor's Perspective: An Empirical Study. *Proceedings of the Pakistan Academy of Sciences: A. Physical and Computational Sciences*, *57*(4), 1–11.

Hanoğlu, E., & Tarhan, A. (2019). An Empirical Study on the Relationship between Open Source Software Success and Test Effort. *2019 4th International Conference on Computer Science and Engineering (UBMK)*, 688–692. https://doi.org/10.1109/UBMK.2019.8907012

Ho, S. Y., & Rai, A. (2017). Continued Voluntary Participation Intention in Firm-Participating Open Source Software Projects. *Information Systems Research*, *28*(3), 603–625. https://doi.org/10.1287/isre.2016.0687

Howison, J., & Crowston, K. (2014). Collaboration Through Open Superposition: A Theory of the Open Source Way. *MIS Quarterly*, *38*(1), 29–50.

Huang, L. K., Lin, Y., & Lin, T. (2005). The nexus between information technology and competitive strategy: A conceptual framework and its hypotheses. *IEEE International Conference on e-Business Engineering (ICEBE'05)*, 171–178. https://doi.org/10.1109/ICEBE.2005.118

Huff, S. L., Maher, P. M., & Munro, M. C. (2006). Information Technology and the Board of Directors: Is There An IT Attention Deficit? *MIS Quarterly Executive*, *5*(2), 55–68.

Hwang, J., Kim, E., & Kim, S. (2009). Factors affecting open technological innovation in open source software companies in Korea. *Innovation: management, policy & practice*, *11*(3), 279–290. https://doi.org/10.5172/impp.11.3.279

Ilgen, D. R., Hollenbeck, J. R., Johnson, M., & Jundt, D. (2005). Teams in Organizations: From Input-Process-Output Models to IMOI Models. *Annu. Rev. Psychol.*, *56*, 517–543. https://doi.org/10.1146/annurev.psych.56.091103.070250

Istiyanto, J. E., Emanuel, A. W. R., & Utara, J. S. (2009). Success Factors of Open Source Software Projects using Datamining Technique. *Proceeding of Information Technology and Communication International Seminar (ITIS)*.

Iwami, S. (2022). A Methodology to Categorize Rapidly Changing Projects of Open Source Software. *2022 Portland International Conference on Management of Engineering and Technology (PICMET)*, 1–8. https://doi.org/10.23919/PICMET53225.2022.9882581

Jacobides, M. G., Knudsen, T., & Augier, M. (2006). Benefiting from innovation: Value creation, value appropriation and the role of industry architectures. *Research Policy*, *35*(8), 1200–1221. https://doi.org/10.1016/j.respol.2006.09.005

Jeppesen, L. B., & Frederiksen, L. (2006). Why Do Users Contribute to Firm-Hosted User Communities? The Case of Computer-Controlled Music Instruments. *Organization Science*, *17*(1), 45–63. https://doi.org/10.1287/orsc.1050.0156

Johannessen, J.-A., Olaisen, J., & Olsen, B. (1999). Strategic use of information technology for increased innovation and performance. *Information Management & Computer Security*, *7*(1), 5–22. https://doi.org/10.1108/09685229910255133

Kane, G. C., & Ransbotham, S. (2016). Content as Community Regulator: The Recursive Relationship Between Consumption and Contribution in Open Collaboration Communities. *Organization Science*, *27*(5), 1258–1274. https://doi.org/10.1287/orsc.2016.1075

Kapitsaki, G. M., Tselikas, N. D., Kyriakou, K.-I. D., & Papoutsoglou, M. (2022). Help me with this: A categorization of open source software problems. *Information and Software Technology*, *152*, 107034. https://doi.org/10.1016/j.infsof.2022.107034

Kawaguchi, S., Garg, P. K., Matsushita, M., & Inoue, K. (2004). MUDABlue: An automatic categorization system for open source repositories. *11th Asia-Pacific Software Engineering Conference*, 184–193. https://doi.org/10.1109/APSEC.2004.69

Koch, S., & Stix, V. (2008). Open Source Project Categorization Based on Growth Rate Analysis and Portfolio Planning Methods. *Open Source Development, Communities and Quality: IFIP 20 th World Computer Congress, Working Group 2.3 on Open Source Software, September 7-10, 2008, Milano, Italy 4*, 375–380.

Kumar, R., Kumar, S., & Tiwari, S. K. (2019). A study of software reliability on big data open source software. *International Journal of System Assurance Engineering and Management*, *10*(2), 242–250. https://doi.org/10.1007/s13198-019-00777-x

Lamastra, C. (2009). Firms' participation in Open Source projects: Which impact on software quality and success.

Larsson, Z. Y., Di Gangi, P. M., & Teigland, R. (2019). Sharing my way to success: A case study on developing entrepreneurial ventures using social capital in an OSS community. *Information and Organization*, *29*(1), 23–40. https://doi.org/10.1016/j.infoandorg.2018.12.001

Lee, J. Y. (2011). *Incremental innovation and radical innovation: The impacts of human, structural, social, and relational capital elements*. Michigan State University. Operations; Sourcing Management.

Lee, S.-Y. T., Kim, H.-W., & Gupta, S. (2009). Measuring open source software success. *Omega*, *37*(2), 426–438. https://doi.org/10.1016/j.omega.2007.05.005

Liao, Z., Deng, L., Fan, X., Zhang, Y., Liu, H., Qi, X., & Zhou, Y. (2018). Empirical Research on the Evaluation Model and Method of Sustainability of the Open Source Ecosystem. *Symmetry*, *10*(12), 1–26. https://doi.org/10.3390/sym10120747

Linåker, J., Rempel, P., Regnell, B., & Mäder, P. (2016). How Firms Adapt and Interact in Open Source Ecosystems: Analyzing Stakeholder Influence and Collaboration Patterns. *Requirements Engineering: Foundation for Software Quality: 22nd International Working Conference, REFSQ*, 63–81.

Lindberg, A., Berente, N., Gaskin, J., & Lyytinen, K. (2016). Coordinating Interdependencies in Online Communities: A Study of an Open Source Software Project. *Information Systems Research*, *27*(4), 751–772. https://doi.org/10.1287/isre.2016.0673

Lindman, J., Paajanen, A., & Rossi, M. (2010). Choosing an Open Source Software License in Commercial Context: A Managerial Perspective. *2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications*, 237–244. https://doi.org/10.1109/SEAA.2010.26

Lingard, L., Albert, M., & Levinson, W. (2008). Grounded theory, mixed methods, and action research. *BMJ*, *337*, 459–461. https://doi.org/10.1136/bmj.39602.690162.47

Long, J., & Yuan, M. J. (2005). Are All Open Source Projects Created Equal? Understanding the Sustainability of Open Source Software Development Model. *Americas Conference on Information Systems (AMCIS)*.

Luo, J. (2022). Data-Driven Innovation: What Is It? *IEEE Transactions on Engineering Management*, *70*(2), 784–790. https://doi.org/10.1109/TEM.2022.3145231

Markus, M. L. (2007). The governance of free/open source software projects: Monolithic, multidimensional, or configurational? *Journal of Management & Governance*, *11*, 151–163. https://doi.org/10.1007/s10997-007-9021-x

Medappa, P. K., & Srivastava, S. C. (2019). Does Superposition Influence the Success of FLOSS Projects? An Examination of Open-Source Software Development by Organizations and Individuals. *Information Systems Research*, *30*(3), 764–786. https://doi.org/10.1287/isre.2018.0829

Midha, V., & Palvia, P. (2012). Factors affecting the success of Open Source Software. *Journal of Systems and Software*, *85*(4), 895–905. https://doi.org/10.1016/j.jss.2011.11.010

Miranda, S. M., Kim, I., & Summers, J. D. (2015). Jamming with Social Media. *MIS Quarterly*, *39*(3), 591–614.

Neufeld, D. J., & Gu, H. (2019). Leadership Emergence and Impact on Open Source Software Project Success: A Comparative Case Study. *Academy of Management Proceedings*, *2019*(1), 11698.

Nolan, R., & McFarlan, F. W. (2005). Information Technology and the Board of Directors. *Harvard Business Review*, *83*(10), 1–11.

Norman, D. A., & Verganti, R. (2014). Incremental and Radical Innovation: Design Research vs. Technology and Meaning Change. *Design Issues*, *30*(1), 78–96. https://doi.org/10.1162/DESI_a_00250

O'Mahony, S., & Bechky, B. A. (2008). Boundary Organizations: Enabling Collaboration among Unexpected Allies. *Administrative Science Quarterly*, *53*(3), 422–459. https://doi.org/10.2189/asqu.53.3.422

O'Mahony, S., & Ferraro, F. (2007). The Emergence of Governance in an Open Source Community. *Academy of Management Journal*, *50*(5), 1079–1106. https://doi.org/10.5465/amj.2007.27169153

Peng, G. (2009). Critical success factors for open source innovation: The case of open source software development. *Issues in Information Systems*, *10*(2), 157–164. https://doi.org/10.48009/2_iis_2009_157-164

Peng, G., Wan, Y., & Woodlock, P. (2013). Network ties and the success of open source software development. *The Journal of Strategic Information Systems*, *22*(4), 269–281. https://doi.org/10.1016/j.jsis.2013.05.001

Piva, E., Rentocchini, F., & Rossi-Lamastra, C. (2012). Is Open Source Software about Innovation? Collaborations with the Open Source Community and Innovation Performance of Software Entrepreneurial Ventures. *Journal of Small Business Management*, *50*(2), 340–364. https://doi.org/10.1111/j.1540-627X.2012.00356.x

Puspitasari, I., & Jie, F. (2020). Making the information technology-business alignment works: A framework of IT-based competitive strategy. *International Journal of Business Information Systems*, *34*(1), 59–82. https://doi.org/10.1504/IJBIS.2020.106796

Radziszewski, P. (2020). Exploring the development of an innovation metric - from hypothesis to initial use. *Journal of Innovation and Entrepreneurship*, *9*(10), 1–29. https://doi.org/10.1186/s13731-020-00118-4

Rahmani, C., Azadmanesh, A. H., & Najjar, L. (2010). A Comparative Analysis of Open Source Software Reliability. *Journal of Software*, *5*(12), 1384–1394. https://doi.org/10.4304/jsw.5.12.1384-1394

Ritala, P., & Hurmelinna-Laukkanen, P. (2013). Incremental and Radical Innovation in Coopetition—The Role of Absorptive Capacity and Appropriability. *Journal of Product Innovation Management*, *30*(1), 154–169. https://doi.org/10.1111/j.1540-5885.2012.00956.x

Rönkkö, M., & Cho, E. (2022). An Updated Guideline for Assessing Discriminant Validity. *Organizational Research Methods*, *25*(1), 6–47. https://doi.org/10.1177/1094428120968614

Sagers, G. (2004). The Influence of Network Governance Factors on Success in Open Source Software Development Projects. *International Conference on Information Systems (ICIS)*, 427–438.

Santos Jr, C. D., Kuk, G., Kon, F., & Suguiura, R. (2011). The Inextricable Role of Organizational Sponsorship for Open Source Sustainability. *Proceedings of SOS 2011: Towards Sustainable Open Source*, 7–11.

Schweik, C. M. (2013). Sustainability in Open Source Software Commons: Lessons Learned from an Empirical Study of SourceForge Projects. *Technology Innovation Management Review*, *3*(1).

Sen, R., Singh, S. S., & Borle, S. (2012). Open source software success: Measures and analysis. *Decision Support Systems*, *52*(2), 364–372. https://doi.org/10.1016/j.dss.2011.09.003

Sen, R., Subramaniam, C., & Nelson, M. L. (2011). Open source software licenses: Strong-copyleft, non-copyleft, or somewhere in between? *Decision support systems*, *52*(1), 199–206. https://doi.org/10.1016/j.dss.2011.07.004

Showole, A., Sahibuddin, S., & Ibrahim, S. (2011). Layered Approach to Open Source Software Development Success. *Communications of the IBIMA*, *2011*, 1–22. https://doi.org/10.5171/2011.160480

Singh, V., Sharma, M., & Pham, H. (2017). Entropy Based Software Reliability Analysis of Multi-Version Open Source Software. *IEEE Transactions on Software Engineering*, *44*(12), 1207–1223. https://doi.org/10.1109/TSE.2017.2766070

Souto, J. E. (2015). Business model innovation and business concept innovation as the context of incremental innovation and radical innovation. *Tourism management*, *51*, 142–155. https://doi.org/10.1016/j.tourman.2015.05.017

Stewart, K. J., Ammeter, A. P., & Maruping, L. M. (2005). A Preliminary Analysis of the Influences of Licensing and Organizational Sponsorship on Success in Open Source Projects. *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, 1–10. https://doi.org/10.1109/HICSS.2005.38

Stewart, K. J., Ammeter, A. P., & Maruping, L. M. (2006). Impacts of License Choice and Organizational Sponsorship on User Interest and Development Activity in Open Source Software Projects. *Information Systems Research*, *17*(2), 126–144. https://doi.org/10.1287/isre.1060.0082

Stewart, K. J., Darcy, D. P., & Daniel, S. L. (2006). Opportunities and Challenges Applying Functional Data Analysis to the Study of Open Source Software Evolution. *Statistical Science*, *21*(2), 167–178. https://doi.org/10.1214/088342306000000141

Subramaniam, C., Sen, R., & Nelson, M. L. (2009). Determinants of open source software project success: A longitudinal study. *Decision Support Systems*, *46*(2), 576–585. https://doi.org/10.1016/j.dss.2008.10.005

Tang, T. Y., Fisher, G. J., & Qualls, W. J. (2021). The effects of inbound open innovation, outbound open innovation, and team role diversity on open source software project performance. *Industrial Marketing Management*, *94*, 216–228. https://doi.org/10.1016/j.indmarman.2021.02.013

Teigland, R., Di Gangi, P. M., Flåten, B.-T., Giovacchini, E., & Pastorino, N. (2014). Balancing on a tightrope: Managing the boundaries of a firm-sponsored OSS community and its impact on innovation and absorptive capacity. *Information and Organization*, *24*(1), 25–47. https://doi.org/10.1016/j.infoandorg.2014.01.001

Thacker, C., & Knutson, C. (2015). Barriers to Initiation of Open Source Software Projects in Libraries. *Code4Lib Journal*, (29).

Tworek, K., Walecka-Jankowska, K., & Zgrzywa-Ziemak, A. (2019). Information Technology Reliability in Shaping Organizational Innovativeness of SMEs. *Organizacija*, *52*(2), 143–154. https://doi.org/10.2478/orga-2019-0010

Vaast, E., Safadi, H., Lapointe, L., & Negoita, B. (2017). Social Media Affordances for Connective Action: An Examination of Microblogging Use During the Gulf of Mexico Oil Spill. *MIS Quarterly*, *41*(4), 1179–1206.

Vargas-Baldrich, S., Linares-Vásquez, M., & Poshyvanyk, D. (2015). Automated Tagging of Software Projects using Bytecode and Dependencies (N). *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 289–294. https://doi.org/10.1109/ASE.2015.38

Walsh, I. (2014). A strategic path to study IT use through users' IT culture and IT needs: A mixed-method grounded theory. *The Journal of Strategic Information Systems*, *23*(2), 146–173. https://doi.org/10.1016/j.jsis.2013.06.001

Walsh, I. (2015). Using quantitative data in mixed-design grounded theory studies: An enhanced path to formal grounded theory in information systems. *European Journal of Information Systems*, *24*(5), 531–557. https://doi.org/10.1057/ejis.2014.23

Wang, L., & Wang, J. (2020). The Impact of Project Initiators on Open Source Software Project Success: Evidence From Emerging Hosting Platform Gitee. *International Journal of Information Systems in the Service Sector (IJISSS)*, *12*(1), 28–43. https://doi.org/10.4018/IJISSS.2020010103

Wang, Q., Jiang, J., & Li, L. (2019). A Reliability Automatic Assessment Framework for Open Source Software. *2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC)*, 91–911. https://doi.org/10.1109/PRDC47002.2019.00033

Wang, T., Wang, H., Yin, G., Yang, C., Li, X., & Zou, P. (2014). Hierarchical Categorization of Open Source Software by Online Profiles. *IEICE TRANSACTIONS on Information and Systems*, *97*(9), 2386–2397. https://doi.org/10.1587/transinf.2014EDP7007

Wang, Y., Wang, L., Hu, H., Jiang, J., Kuang, H., & Tao, X. (2022). The Influence of Sponsorship on Open-Source Software Developers' Activities on GitHub. *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*, 924–933. https://doi.org/10.1109/COMPSAC54236.2022.00144

Wasson, C. R. (1960). What is "New" about a New Product? *Journal of Marketing*, *25*(1), 52–56. https://doi.org/10.1177/002224296002500109

Wen, W., Forman, C., & Graham, S. J. (2013). Research Note—The Impact of Intellectual Property Rights Enforcement on Open Source Software Project Success. *Information Systems Research*, *24*(4), 1131–1146. https://doi.org/10.1287/isre.2013.0479

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

West, J., & O'mahony, S. (2008). The Role of Participation Architecture in Growing Sponsored Open Source Communities. *Industry and Innovation*, *15*(2), 145–168. https://doi.org/10.1080/13662710801970142

Wisdom, J. P., Chor, K. H. B., Hoagwood, K. E., & Horwitz, S. M. (2014). Innovation Adoption: A Review of Theories and Constructs. *Administration and Policy in Mental Health and Mental Health Services Research*, *41*, 480–502. https://doi.org/10.1007/s10488-013-0486-4

Wu, J., Huang, X., & Wang, B. (2023). Social-technical network effects in open source software communities: Understanding the impacts of dependency networks on project success. *Information Technology & People*, *36*(2), 895–915. https://doi.org/10.1108/ITP-09-2021-0684

Wu, J., Goh, K.-Y., Li, H., Luo, C., & Zheng, H. (2016). The Effects of Communication Patterns on the Success of open Source Software Projects: An Empirical Analysis from Social Network Perspectives. *Journal of Global Information Management (JGIM)*, *24*(4), 22–44. https://doi.org/10.4018/JGIM.2016100102

Ye, Y., Nakakoji, K., Yamamoto, Y., & Kishida, K. (2008). The Co-Evolution of Systems and Communities in Free and Open Source Software Development. In *Global information technologies: Concepts, methodologies, tools, and applications* (pp. 3765–3776). Igi Global.

Yılmaz, N., & Tarhan, A. (2022). Quality evaluation models or frameworks for open source software: A systematic literature review. *Journal of Software: Evolution and Process*, *34*(6), 1–34. https://doi.org/10.1002/smr.2458

Zanella, G., & Liu, C. Z. (2020). A Social Network Perspective on the Success of Open Source Software: The Case of R Packages. *Proceedings of the 53rd Hawaii International Conference on System Sciences*, 471–480.