Exploring the effects of temporal evolution in open source software projects

Information Systems Science
Master's thesis

Author:
Sven de Waard

Supervisor:
Phd. Poonacha Medappa

30.07.2023
Dordrecht, the Netherlands

Master's thesis

Subject: Open source software development
Author(s): Sven de Waard
Title: Exploring the effects of temporal evolution in open source software projects
Supervisor(s): Poonacha Medappa
Number of pages: 70 pages + appendices 3 pages
Date: 30.07.2023

Key words: keyword 1, keyword 2.

This study addresses the governance and coordination challenges faced in open source software (OSS) development projects, a modern development approach that has seen rapid adoption in recent years. Unlike traditional software development, OSS projects involve a geographically dispersed base of volunteers working on the code openly, usually without formal hierarchies or contracts. As a result, OSS projects can face scalability issues, such as developers freely abandoning projects or disputes leading to project forking. The study of OSS governance and its underlying mechanisms has seen recent interest. Some of this research has proposed that social dynamics and community structures may change as a result of evolution and growth. This study therefore is an attempt to understand how different types of OSS projects evolve over time. The goal of this study is to identify evolutionary patterns in the community's approach regarding the trade-off between innovation and sustainability. The research question to be answered is as follows: *What evolutionary patterns can be identified with regards to the community and its approach to innovation and sustainability of open source software development projects?*

To answer this question, a quantitative research has been carried out based on the data of over 1,500 open source software projects created by Google, Microsoft, or Apache. This data has been retrieved through the GraphQL API of GitHub – the world's largest open source development platform. With the help of Python scripts, this data has been analyzed to identify certain phenomena – patterns – in the evolution of OSS projects.

Based on the academic literature, a framework is developed to categorize OSS projects, emphasizing the trade-off between innovation and sustainability. The results indicate that after the first project release, the attention towards smaller features increases. Innovation and sustainability levels are however not affected. Over time, projects with high innovation levels tend to transition towards a more defensive approach. Simultaneously, projects with initially low sustainability levels show improvement and reach sustainable levels after the first year. Notably, the size of the community is a key predictor for new contributor inflow, while having more pull request reviewers proves effective in both contributor retention and innovation. Interestingly, contributors tend to remain engaged for longer periods when involved in non-profit sponsored projects compared to for-profit sponsored projects. Moreover, a change in platform ownership does not significantly impact other organizations within the platform. Lastly, the study reveals that early contributors show longer retention, whereas the inflow of new contributors gradually decreases as OSS projects age.

These are the identified evolutionary patterns in open source software projects and show that while there are inherent differences between such projects, they do commonly follow the same or similar events. The degree of change is dependent on organizational and project characteristics. As this research is focused on solely sponsored open source projects, further research could focus on examining community-founded projects and comparing them with the results of this study. Additionally, investigating the relationship between contributor turnover, innovation, and project sustainability would be valuable.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Software development is traditionally governed by a formalized structure and control processes, based on the hierarchy within a project or firm. Developers work on issues and are being audited by lead developers and/or managers. This is what Raymond (1999) would describe as a "cathedral", a place where software is carefully crafted by individuals – a few "wizards or small bands of mages" - within an isolated team. Source code is not to be seen until it is released, and it is often characterized by formality and top-down hierarchies.

Nowadays, many argue for a less traditional and more open approach: open source software (further mentioned as OSS) development. This is an approach that has seen rapid diffusion over the years, with projects like Apache Web Server and Linux growing more than 200% annually (Lerner & Tirole, 2002). Furthermore, it has seen significant capital investments, and "traditional" organizations are increasingly interested in participating in the movement. As a result of this, there have been numerous successful and mainstream OSS projects (Hogarth & Turner, 2005; Nelson et al., 2006).

OSS development is described by Raymond (1999) as a "great babbling bazaar of differing agendas and approaches". The bazaar is a kind of community binding together developers that are spread out geographically (Lerner & Tirole, 2002). They are connected through a shared purpose of sharing code to develop and refine programs in order to address a practical need or solve a common problem (Still, 2008). As opposed to proprietary development, OSS is built on rapid iteration, open development processes, and a place where errors are not kept secret (Hogarth & Turner, 2005). In addition, OSS development is characterized by its source code being openly accessible to any user, who has the right to read and even modify it. Therefore, Ye et al. (2005) argue that all users are potential developers who may or may not join the community in developing or maintaining a software product.

Such characteristics have implications on the way that projects – and especially people – should be governed. In addition to the problems faced by proprietary software development, you are now also dealing with the complexity of coordinating efforts of a geographically dispersed base of volunteers (Nelson et al., 2006). Pay attention to that last word – it is commonly argued to be the base of many of OSS's scalability issues.

Most developers are not paid to work on a project and are not under the bindings of a contract. Therefore, they are free to abandon a project (or move onto an alternative project) whenever they want. This commonly occurs on projects that lose popularity and therefore may gradually lose its developer base until its ultimate discontinuation (De Noni et al., 2013; Lerner & Tirole, 2002; Ye et al., 2005). Another issue that may occur, is the "forking" of projects. This is a phenomenon in which a developer takes the original source code, merges it to their own repository (development base) and starts a "spin-off", or alternative variant, of the software. Forking can be caused by disputes in the community, for example over product design (Lerner & Tirole, 2002; Nelson et al., 2006) or leadership (Ye et al., 2005).

These issues have led to an increase in the specific study of OSS governance and its mechanisms. Many of these studies are focused on trying to identify the elements of OSS governance and how it differs from traditional governance. These elements are then used to classify projects into certain community structures and their corresponding direction, control, and coordination efforts (de Laat, 2007; Ye et al., 2005). A mechanism commonly mentioned in the literature is *innovation* and its trade-off to *sustainability* (Nolan & McFarlan, 2005; Ye et al., 2005). Innovation is the implementation of something novel, which can be approached on a spectrum between radical (offensive) innovation and incremental (defensive) innovation (de Jong & Vermeulen, 2003). The mechanism at place here is that the more radical the innovation is, the harder it becomes to properly coordinate the efforts of all individuals, compromising sustainability. At the other end, a defensive innovation approach is much easier to manage and therefore you are able to offer stable services to all stakeholders. Thus, a community has to decide what approach fits the project best, and whether to focus on sustainability or innovation, or to find a place in between.

Some have led to the proposition that as OSS communities grow and prosper, the social dynamics and structure of the community (and thus coordination mechanisms) may change. For example, a recent study by Ferraz and dos Santos (2022) came to a conclusion that these mechanisms are quite flexible. They even argue that the common definition of cathedral versus bazaar is actually dynamic and may change over time. In addition, Ye et al. (2005) propose the theory that an OSS community (adapted from F/OSS) transforms through co-evolution, where contributing members and OSS systems are both sources of each other's evolution. The authors also mention that OSS systems and communities may co-evolve differently depending on the goal of the system and the structure of the community.

These studies are all qualitative and based on a select few projects. According to Nelson et al. (2006), a significant opportunity exists for studying the evolution of coordination mechanisms in OSS projects. Additionally, Markus (2007) questions whether variations in governance are accidents of history or corresponding to differences in the community [context or nature] of OSS projects. Markus also adds that it is not yet clear to what extent these variations are consciously designed or rather organically evolved.

It is clear that there is a research gap in the study of how different types of OSS projects evolve over time, and what implications to the coordination of its community follow. This study therefore aims to answer the following research question:

> RQ: **What evolutionary patterns can be identified with regards to the community and its approach to innovation and sustainability of open source software development projects?**

This question is approached according to the following sub-topics:

- Identifying a theoretical framework for categorizing OSS projects
  To understand evolutionary patterns, first must be noted that OSS projects are quite different. Hence the need for examining these differences and shaping them into a categorization framework.

- Evolutionary dynamics of an OSS project: going open source
  The first official release of an OSS project is an important one, since it denotes the moment of "going open source". It is an event (and thus potential change) that most OSS projects go through.

- Project characteristics and their effects on evolution
  By categorizing projects based on the identified framework, the evolution can be measured over time. Additionally, the underlying characteristics must be studied to identify causes for change.

- Organizational characteristics and the effects of platform ownership
  OSS projects can be sponsored by a formal organization, and therefore it is important to study the effects over time of such sponsorship based on the characteristics of the organization. Furthermore, a change in platform ownership may have additional effects on other organizations as well.

- Contributor characteristics and their connection to the community
  Contributors are the vital building blocks of an OSS project, and thus it is important to study the changes of contributor acquisition and retention over time.

In short, this research is an attempt at identifying OSS projects, events that may happen during their lifetime, and then determining how they change over time; ultimately shaping this into common patterns. This has significant practical implications on the governance and management of such projects. By identifying how projects commonly change, why they change, and what is then necessary to properly manage the development and community, it should make it easier for organizations to prepare for a shift towards OSS development, or to rethink their approach to innovation and/or sustainability. Furthermore, existing projects looking to make the next step can both be better prepared for newly arising issues and be able to create a sustainable environment that continuously attracts new contributors (De Noni et al., 2013).

Firstly, the literature will be studied to try to identify what different types of projects exist and according to what dimensions they differ. Then, the actual research is conducted using data analysis in Python. The data for this research has been collected through GitHub's GraphQL API using several Python scripts. This extensive process resulted in commit and pull request data for 1,785 projects by either Microsoft, Google, or Apache. The final dataset includes details of well over 1,000,000 commits. After the results have been collected using this dataset, they will be combined with the literature to discuss the findings and their implications for both the academic and practical fields surrounding OSS development.

2 LITERATURE REVIEW

2.1 A brief introduction to open source

In order to fully comprehend the thoughts and processes behind the open source movement, it is important to know its definition, characteristics, and origins. Literature on this topic dates back to somewhere in the late 1990s. While authors such as Raymond (1999) have defined OSS development as a reflection of a real-life community (the bazaar versus traditional cathedral development), an official definition exists. Founded in 1998, the Open Source Initiative (OSI) actively campaigns for the open source movement and maintains adherence to its definition (Still, 2008). Their mission is to educate about and advocate for the benefits of open source, for example during conferences and events (Open Source Initiative, 2023). The non-profit organization aimed at distinguishing this particular form of development from "free software" and thus coined it "open source". The term was adopted quite quickly, with software such as Linux, sendmail, and Python supporting it, resulting in an increased interest and participation in the movement (Open Source Initiative, 2018).

What is commonly believed is that the label of open source means having access to the source code. While that is indeed one of its characteristics, it is not enough to be able to mark a project as such. The software must comply with some criteria to be allowed to use the trademark (Open Source Initiative, 2007; Still, 2008). In short, a project must allow any party to create and share copies ("derived work") of the software for free. Furthermore, the source code can be freely viewed and even modified by any user, if they wish to do so. Lastly, its license must apply to all parties and derived works.

Proprietary software companies often do not follow such criteria in their development processes. They close access to their source code, because it is considered to be intellectual property (Still, 2008). Furthermore, if users can change the source code of a paid piece of software, they would not have to pay for any future version; they could even redistribute it for a lower price (or for free). In OSS, ownership and profitability of intellectual property is not such an issue. The software is a product of collective effort – a peer production of software if you will – and therefore it belongs to the community in which everyone shares a part (de Laat, 2007; Hogwart & Turner, 2005).

2.2 OSS communities

It is clear that the philosophy of OSS is built on development communities. Therefore, to be able to classify projects, one has to know the characteristics of the community building and maintaining the software. From the introduction it seems that communities develop themselves organically, and most often consist of volunteers who are organizationally and geographically dispersed (Lerner & Tirole, 2002; Wei et al., 2017). According to Ye and his colleagues (2005), communities of practice are formed by a group of people who are informally bonded by their common interest and shared practice in a specific domain. From this, one could say that a community is originated from a certain problem or interest area, and that may attract people who share it to join the project. This is supported by Raymond (1999), who states that projects are started by individual vision and brilliance (someone has a solution for some issue) and is then amplified through constructing such communities. Lerner and Tirole (2002) explain this further by stating that individual solutions may become general solutions for wide classes of users. This is called "user-driven need" by Schweik (2013). The same article explains that a *social capital* is built between the developers and their user base, which in turn grows their user base. As a community grows, more progress is made on the software, and more users may be attracted to become a developer themselves. Just like Ye and his colleagues (2005) mentioned, all users can be seen as potential developers, who further improve the software, attract more people… – and that is exactly how an OSS development community grows organically.

Ye et al. (2005) make it clear that an associated community is vital for an OSS project. Without it, it is unlikely to sustain long term success. Therefore, if a project is not picked up by the general public – perhaps because the issue is too specific to the original developer(s) – it is not expected to last a long time. In addition, if the project stops growing because it cannot attract enough developers, it also tends to die. In other words: the success of an OSS project is related to the growth in the size of the community (Schweik, 2023; Ye et al., 2005).

2.2.1 Contributors

A community consists of people and is reliant on them for its sustainable growth. Therefore, it is important to study their characteristics and motivations for collaborating.

According to de Laat (2007), academic literature on OSS has historically seen three phases, the first of which is focused on the backgrounds and motivations of people volunteering to contribute. From these studies can be concluded that the volunteers are usually between 30 and 50, have ICT qualifications to some degree, and have paid jobs during which they are commonly allowed to work on OSS projects.

Motivations for contributing to projects seem to be both internal and external, and span across economic, social, and technological areas (Feller & Fitzgerald, 2002). For example, many may find working on an OSS project to be much "cooler" than the routine tasks they are performing for their paid jobs (de Laat, 2007; Lerner & Tirole, 2002). This is called *intellectual stimulus* and shows that volunteers are often just looking for a challenge. At the same time, this is an enabler of one of the main issues of open source development: the dependency on popularity (Lerner & Tirole, 2002). As people volunteer to contribute to a project, they are free to leave at any point in time, especially since the switching costs are low. Once a project is no longer deemed as "challenging", they will both lose contributors and attract fewer users to become developers (De Noni et al., 2013; Ye et al., 2005). This shows the importance of continuously trying to innovate and attract people by expressing the challenges that are still open. Another internal motivation that has been explored earlier, is that the software they are working on can solve one of their own problems. People will help develop the software that they will use for their own purposes once it is completed (to the extent that they need it to be) (Nelson et al., 2006; Ye et al., 2005). The same could apply to those that are looking for a solution to help them achieve a goal for the organization they work for. Instead of "wasting" time by not focusing on the mission, they may gain an increase in performance if the OSS solution is implemented within the firm.

For those that are not currently employed, developing OSS may lead to future job offers, reflecting a career concern incentive (Lerner & Tirole, 2002; Nelson et al., 2006). Once your work is gaining positive attention from the community or its leadership, you may be offered to work for other (also non-OSS) projects. It is increasingly common for OSS communities to pay their developers for their work, although this is often less than a regular salary. This again reflects the reason why many OSS developers work on these projects next to their jobs. In addition, some deliberately choose OSS just because they believe that source code should be open (de Laat, 2007). Lastly, an ego gratification incentive may occur, wherein one has a desire for recognition by his peers. This is a volunteer who cares about reputation and wishes to share his work through OSS in order to achieve this (Lerner & Tirole, 2002; Nelson et al., 2006).

But as most studies have focused on the reason *why* people contribute to OSS projects, Markus (2007) rightfully points out that it is also relevant to know the reasons for choosing *which* project to contribute to. There are over a hundred million repositories on GitHub alone, the most popular and preferred platform of OSS developers (AlMarzouq et al., 2022). This makes it not easy for someone new to the ecosystem (or a switching developer) to pick a project to contribute to. GitHub does provide a "trending" page, showing the top repositories that have gained the most traction today (Trending Repositories on GitHub Today, n.d.), but this is an obvious result of multiple developers choosing a specific project in the same time period. Markus (2007) argues that the governance mechanisms at use in an OSS community may have inherent motivational potential, which can determine either *what* project to contribute to, or *how much effort* to contribute to the selected project. This is based on the *organizational climate* – which can thus have a positive or negative impact on the supply of new developers. The organizational climate can be defined by the quality of the social atmosphere and of the social relationships in the community (De Noni et al., 2013). Therefore, individuals, given the choice, favour working in an environment that aligns with their interests, attitudes, and skills. From Markus' study (2007), contributors commonly showed a preference for "open" OSS projects, compared to "gated" communities. Open communities are those that reflect democratic policies and is based on the perceived opportunities for participation (similar to the "open challenges" mentioned earlier), a transparency in the governance arrangements, and the accountability of decision makers. Gated communities on the other hand are those with restricted licenses, or also commonly named "sponsored" (De Noni et al., 2013). A single firm, or a coalition of firms, has significant control over governance in a sponsored community.

This demonstrates that open communities seem more inviting and challenging to potential developers compared to closed ones. But in addition to this, the content or nature of the project may also be a determining factor in deciding what community to join. For example, projects built on leading-edge technologies tend to attract people willing to learn and experience them (De Noni et al., 2013). Such projects often come with complex challenges. Moreover, those people often value their freedom and want to create without having to ask permission – a characteristic of an open community. In contrast, projects that are merely focused on efficiency or standardization are unlikely to gain a large following. They are either not challenging enough or are not built on anything “new and interesting”.

2.2.2 Community structure and roles

Now that it is clear what communities are and why people voluntarily join their causes, it is important to know what hierarchies exists among them (if any), how new people can write their first contribution, and how they can move up the ranks like in a “regular” firm.

In a community of practice, there is not usually a formally defined hierarchy (Ye et al., 2005). Even though leadership is often well defined (Ferraz & dos Santos, 2002), other members of the community are not assigned certain roles by someone else. In fact, they assume roles themselves based on their individual effort, skills, and interests. Jin et al. (2007) add to this by stating that as a community grows because of an increased interest in the software, it differentiates itself into various roles. This process has been labeled the “joining script”, as defined by Krogh and his colleagues (2003). It specifies the path a user follows from the first moment he or she finds out about a piece of OSS. The proposition is that if someone follows this script, they are more likely to be granted access to the community than those who do not. Ye et al. (2005) have attempted to specify a common joining script, which will be explained in the following section.

New members usually start as *passive users* and are similar to the users of proprietary software. They may be attracted to the software because it solves a problem they are having. Then, they either start reporting a few bugs (errors, flaws in the software) they find whilst using the software (*bug reporters*), or they may even start reading the system to learn more about it (*readers*). After working with the software for some time, a reader or bug reporter may want to start fixing the bugs they or others found – or they could add a whole new twist to the system to increase its capabilities or performance according to their own needs. When their work is made public and benefited from by other members of the community, they could be recognized as *peripheral developers*. This is when they make their first code contribution to the project. In the steps forward, their role is determined by the recognition they earn through the quantity and quality of their contributions. An *active developer* is one that regularly contributes to the project and belongs to one of the major development forces. They work closely together with the *core members* and *project leader(s)*. The core group is responsible for guidance and coordination of the project's development. They have usually been actively involved in the project for a long time and are often seen as an almost elitist group. The possibility of joining this group can be low, but that highly depends on the project's existing structure and its goals.

This core group is recognized by many different authors studying OSS communities (Eseryel et al., 2020; Ferraz & dos Santos, 2022; Lerner & Tirole, 2002; Setia et al., 2012; Ye et al., 2005), although sometimes given different names. In addition, many agree that the decision-making power increases as you move more towards the core. This is best visible in figure 1, which also shows the relative size of each of the previously discussed classes (*stakeholders* are not members of the community; they only rely on the implementation of OSS software for the services they are using).

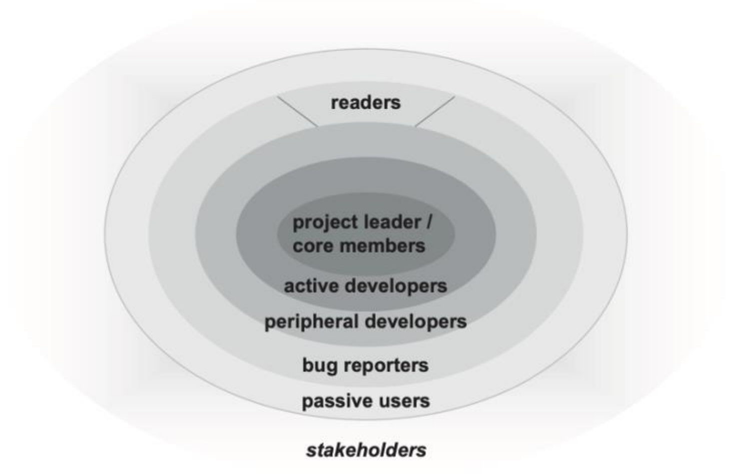


Figure 1 General structure of an OSS community (Ye et al., 2005)

A well-balanced developer community is essential for the success of an OSS project. The core group should be a relatively small group of people that together are responsible for about 80% (or more) of new functionality (Ye et al., 2005) – this is also recognized by Kaur and Chahal (2022) and de Laat (2007). Secondly, a larger group should be present for fixing bugs. Lastly, an even larger group is needed for testing and reporting issues. Lerner and Tirole (2002) found that among nearly 13.000 unique OSS contributors, about 75% made only one contribution. Those that make more than 5 contributions are less than 1 in 25. However, the top 20% of contributors seemed to account for 81% of total code contributions. This is in line with the "balanced community" description by Ye and his colleagues (2005) and gives a number of how large this group should be.

The reason why it is important to have a large group of peripheral developers – those that commit irregularly – is because they help a project to grow (Kaur & Chahal, 2022). Together, they form a pool from where core members may ascend. This means that if there are little developers in the peripheral group and some of the core members decide to abandon the project, there is a significant possibility that the project will not survive. Therefore, both the *maintainability* (bug fixing) and *sustainability* (steady base of developers) depend on this class of people. From this section, it seems vital that both the attraction and retention of contributors is continuously stable in order to sustain development and growth.

2.2.3 Contributor retention, abandonment, and turnover

Contributor retention is also a topic that was studied in the paper by Kaur & Chahal (2022). It refers to the time a member stays involved with the project since his first contribution. The study shows that new-comers are more likely to abandon the project than those who joined some time ago. This is in line with the 75% of members who only made one contribution in the study by Lerner and Tirole (2002). One way to counter this, is to improve their engagement in the project (Kaur & Chahal, 2022). Onboarding processes, improving their motivation, and a more responsive community are mentioned as methods to tackle this issue. Furthermore, the authors mention that abandonment is higher among non-core members than in the core group. Providing the non-core group with some coding tasks besides their maintenance tasks might help increase their overall retention. All of these factors are means to achieve a higher sense of belonging to the community, which is a social determinant of loyalty.

Iaffaldano et al. (2019) propose an additional state between an active contributor and his abandonment. Instead of just looking at retention from first to last contribution, they consider the *sleeping* developers who are merely taking a break from contributing code to the project. They make a division between *sleeping* and *dead* developers, where the latter is a permanent break from the project (abandonment). One of their interviewees argued that a developer can be considered *dead* if they show absolute inactivity in the last year. A *sleeping* developer can still show activity in the project in other areas than code contribution, such as pull request reviews or participation in forum discussions. As soon as the motivation for these developers to continue contributing returns, they cancel their break and become active again. For example, one may feel their knowledge on a specific issue or feature is required to progress the project. Another reason could be that a change in the project structure (such as governance or leadership) triggers their return.

One issue with this *sleeping* state is that it is hard to determine or measure. Consider the following situation:

> a developer has contributed to the project for almost a year before he suddenly decides to stop. No activity has been sighted from his for two months after his last commit (code contribution). Can he now be considered *dead*? According to one of the interviewees in the discussed paper, he cannot until his last contribution is at least one year ago. But in the meantime – how can the community know if he is ever planning on returning, after his supposed break? Does retention continue in this period of time?

This is why *sleeping* can be a temporary condition that ultimately leads to the developer's death. After not being a part of the project for a long time, they may lose interest, awareness, or even knowledge. For projects that are very active (e.g., many developers and continuous innovation), this may all happen faster until the situation where it nearly feels like the developer has to start the onboarding again.

Rashid et al. (2019) have studied this specific phenomenon of knowledge loss in OSS projects. Contributors gain experience and skills, as well as project related knowledge, as they continue working on the project. As soon as they abandon the project (or take a break for that matter), this valuable knowledge is lost if not shared with others. Especially if the abandoned developer was responsible for a specific feature and its underlying code, software development could halt. There are of course ways to deal with such issues, like knowledge management in the shape of code comments or a strong sharing social climate, but that does not take away the fact that a high turnover is detrimental to an OSS project's stability. Even though project turnover is sometimes considered helpful for the vitality of an OSS project due to fresh energy, ideas, and thus innovation (Ye et al., 2005), it may disrupt the community productivity and the product's quality. This is due to the issue of both regaining lost knowledge and sharing knowledge with those recently joining the project. Interestingly, this is similar to the phenomena described by Brooks' Law (Brooks & Brooks, 1975), wherein adding manpower to a late software project makes it later due to the "ramp up" time taken away from active development. Even though the OSS project is probably not considered "late", its innovation may be slowed down if there are high levels of turnover.

In this chapter, the most important aspects of the community and its contributors have been discussed. It is now clear why people volunteer to contribute and how they decide on a project to join. The project's climate could be an interesting dimension for determining the type of an OSS project. In addition, the joining script could be useful for explaining certain phenomena that might be found in the data. Lastly, it is clear that both the balance in contributors and a continuous intake of new people are vital for a project's sustainability.

2.3 OSS community governance & management

Governance in open source is defined as

> the means of achieving the direction, control, and coordination of wholly or partially autonomous individuals and organizations on behalf of the OSS development project to which they jointly contribute (Markus, 2007)

The definition places an emphasis on the governance of individuals and the project to which they jointly contribute. This is a commonly accepted definition that has been widely used among researchers studying OSS governance (De Noni et al., 2013; Di Tullio & Staples, 2013). However, researchers have different views on the purpose (goals) of OSS governance. According to Markus (2007), there are three main positions. The first position views OSS governance as the solution to *collective action dilemmas*, that is, to provide an incentive for participation. An example of this is the control over the ownership of the software or benefits that arise from the OSS project. This view is similar to the first stream of research identified by de Laat (2007), spontaneous governance. In the second position, OSS governance is seen as a solution to the *routine challenges of coordinating interdependence*. Governance is then concerned with how high quality products can be created from individual contributions and the complexity of coordinating them. The third view, that was relatively underdeveloped back in 2007, sees OSS governance as a motivation for its members. By providing both private benefits (like a salary or reputation) and a positive *organizational climate*, their motivation to contribute more effort to the project increases. In fact, some studies argue that a positive climate may even be more effective than private benefits (Markus, 2007).

2.3.1 Leadership

As discussed earlier, de Laat (2007) explores three different streams of research based on specific OSS beliefs. The second phase identified, was that of "internal governance". It counters the argument that communities *spontaneously* create a coordination and control model based on the differences among individual contributions. It is said that projects that have existed for a longer time employ a wide range of formal tools for internal governance. One of these tools is the *delegation of decision-making*. OSS communities are built on all sorts of decisions, such as methods used and release structure (Markus, 2007). De Laat (2007) argues that the most important decision is that of who decides what code changes get included into the product. GitHub explains this process as "merging a pull request into the main branch" (Merging a Pull Request - GitHub Docs, n.d.), and can only be done by those with push (write) access to the repository. In many projects, there is a specific group of people that can review pull requests before they are either merged or sent to the person who can merge it for them; other projects work with voting systems via discussion boards. Alami et al. (2021) add to this by distinguishing three governance styles for the evaluation of pull requests: *protective, equitable,* and *lenient*. In a *protective* style, the project leader and his subordinates have absolute power, thus requiring commitment to win the "gatekeeper's" trust. In an *equitable* style, governance is simply based on fairness, a balanced and technically grounded decision irrespective of who contributes. Lastly, a *lenient* style is one in which a tolerant and positive environment is more important than building error-free software. Such communities believe that creating enthusiasm and mentoring contributors are leverages for the benefit of the entire community. *Equitable* and *lenient* styles are examples of *decentralized* designs for decision-making. In a *centralized* design, only the project leader(s) can decide what code to include, similar to the *protective* evaluation of pull requests.

Even though there are decentralized designs, leadership is often well defined as was found earlier in this paper (Ferraz & Dos Santos, 2022). For smaller communities, leadership is often realized by the person or group that started the project (de Laat, 2007). This approach is viable because the leader can personally coordinate the efforts of the individual members. In larger projects, leadership can be based on *autocracy* or *democracy.* In an autocratic regime, the leader is self-appointed and is probably still the person who initiated the project. In a democratic community however, the leader is chosen by means of regular electoral processes. A lot of trust is put in the community to decide on a capable leader in this design.

According to Di Tullio and Staples (2013), it is also possible for the core members of the community to form a group that has coordination and control responsibilities (effectively, leadership). In that case, they are the ones who decide on a member to join the core group, instead of it being purely based on those that committed the most to the project. Furthermore, they could set some rules and sanctions to control the behaviors of individual members. It is a different perspective and takes on a vertically centralized but not always autocratic approach. The members of the core group vote on the decisions they have to make, so there is no self-appointed leader in this situation. If a member of the group does not comply, they can simply get "relegated". Early on in a project, this group may be quite small. However, as the project grows in size, more developers will get involved, thus spreading the coordination and control structures across more individuals (Eseryel et al., 2020).

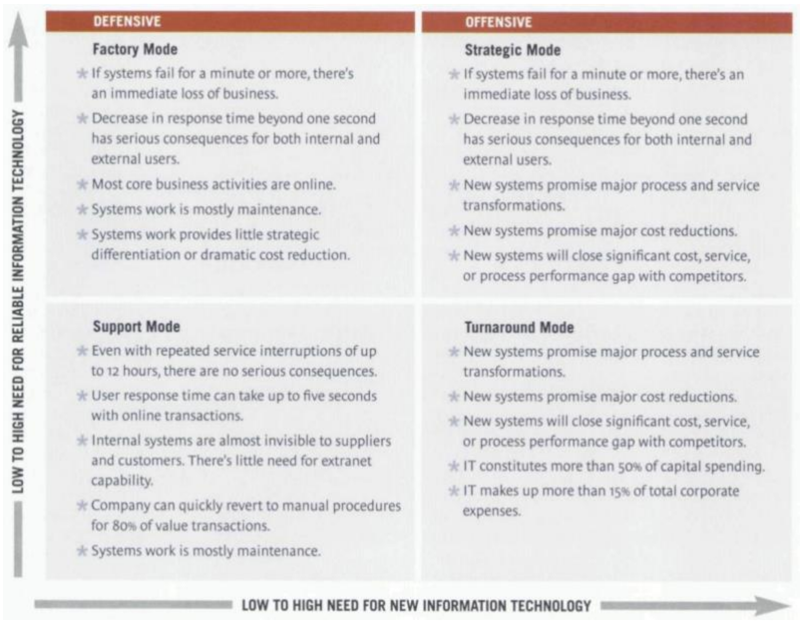
2.3.2 Organizational ownership & externalization

The third phase of OSS research discovered by de Laat (2007) was that of “governance towards outside parties”. This is one of the first research areas in OSS that deliberately accounts for the evolution or transformation of a community. Studies identified that as projects grew larger and thus achieved more success (which are related according to Ye et al. (2005)), the outside world or external parties could no longer be ignored. As organizations are increasingly interested in OSS and want to join in on the movement, the core values of the movement are in danger. This is especially true when the hierarchy and opacity of the mother firm are being transferred to the newly created OSS projects. In addition, OSS developers are increasingly in danger of infringing upon software patents, as patents are created more and more. This means that lawsuits are more likely to occur, and de Laat (2007) even fears for the collapse of the OSS “experiment” as a whole.

As a means of preventing or dealing with such issues, many OSS projects have created a non-profit foundation as a sort of “legal shell” (Feller et al., 2007). It is meant to enable the collaboration between the community and outside, corporate actors. Furthermore, the foundation commonly holds the property rights to the software created by the community, in addition to things like copyright licenses, brand names, and trademarks (de Laat, 2007). However, everything related to the community and its structure – norms, (internal) governance, culture, and project management areas such as requirements planning – are separated from the foundation. Only in rare cases do the foundations interfere with the actual development of the software product, which could be a forebode to an OSS project transforming from an organic form to reflecting a hierarchical firm (de Laat, 2007). But for the most part, the foundation is there to protect and represent the project. Every outside party that wishes to sponsor the project (and that may want to have a say in it) has to go through the foundation first.

2.4 Framework for categorizing OSS projects

To develop a framework for categorizing OSS projects, it is useful to stake a step back and see how software projects in general can be classified. The IT Strategic Impact Grid by Nolan and McFarlan (2005) is a good example that describes two strategic issues for deciding how much impact an IT system has on the organization. The first issue is about "*how much the company relies on effective, uninterrupted, secure, smoothly operating technology systems*". This separates the systems that a firm's continuity relies on from those that do not disrupt the business when issues occur. The second issue is about "*how much the company relies on IT for its competitive edge through systems that provide new value-added services and products or high responsiveness to customers*". This issue seems to be more about innovation – bringing something new to the customer – rather than software stability and business sustainability. Two IT governance strategies are derived from this: *defensive* IT and *offensive* IT. The authors describe that projects usually initiate as an offensive IT solution, bringing innovation and strategic advantage, but in the long run end up as defensive IT that the core business relies upon. This is an interesting temporal change that should be tested for OSS projects. In total, firms can find themselves in one of the four quadrants of the IT Strategic Impact Grid (see figure 2).


DEFENSIVE

Factory Mode
- If systems fail for a minute or more, there's an immediate loss of business.
- Decrease in response time beyond one second has serious consequences for both internal and external users.
- Most core business activities are online.
- Systems work is mostly maintenance.
- Systems work provides little strategic differentiation or dramatic cost reduction.

OFFENSIVE

Strategic Mode
- If systems fail for a minute or more, there's an immediate loss of business.
- Decrease in response time beyond one second has serious consequences for both internal and external users.
- New systems promise major process and service transformations.
- New systems promise major cost reductions.
- New systems will close significant cost, service, or process performance gap with competitors.

Support Mode
- Even with repeated service interruptions of up to 12 hours, there are no serious consequences.
- User response time can take up to five seconds with online transactions.
- Internal systems are almost invisible to suppliers and customers. There's little need for extranet capability.
- Company can quickly revert to manual procedures for 80% of value transactions.
- Systems work is mostly maintenance.

Turnaround Mode
- New systems promise major process and service transformations.
- New systems promise major cost reductions.
- New systems will close significant cost, service, or process performance gap with competitors.
- IT constitutes more than 50% of capital spending.
- IT makes up more than 15% of total corporate expenses.

LOW TO HIGH NEED FOR RELIABLE INFORMATION TECHNOLOGY
LOW TO HIGH NEED FOR NEW INFORMATION TECHNOLOGY

Figure 2 The IT Strategic Impact Grid by Nolan & McFarlan (2005)

This model describes the impact of using IT systems from the perspective of the organization. Unfortunately, it cannot be directly applied to OSS since those are separate systems that this study is viewing from the perspective of its public community. Furthermore, this study does not focus on the implementation of such systems. However, the issues that describe the two dimensions of this model may be of interest for developing an OSS categorization framework. Questions regarding the innovative nature and expected stability of the system (viewed on its own, separated from a firm that may rely on it) are two ways of classifying a repository.

Ye et al. (2005) propose a similar division of OSS projects. While not describing it along the two dimensions of the IT Strategic Impact Grid, they do take into account similar issues. Three different types of F/OSS (Free/Open-Source Software, a different terminology that is perhaps more specific than OSS) are identified:

1. *Exploration-oriented F/OSS*. These are the types of projects that push the frontier of software development, e.g., high (offensive) levels of innovation. Such projects are built by expert programmers, usually led by the hand of a project leader. It is not easy to get into the core developer group. As exploration is the main factor here, it can be assumed that the system is not particularly expected to last.

2. *Utility-oriented F/OSS*. These projects are built for “scratching a personal itch” (Raymond, 1999), or answering a personal developer’s needs. A working piece of software is more important than being “perfect” or stable. There are high chances of forking for such projects, since the solution is usually picked up and adapted by others with similar issues. But as seen earlier, a shared issue could also be the cause of a growing community, thus ultimately changing the governance needs of the project.

3. *Service-oriented F/OSS*. Lastly, service-oriented software is built on the premise of providing stable and robust services to all stakeholders. Innovation can only be implemented after carefully being considered, to not disrupt its services. When compared to the IT Strategic Impact Grid, this seems similar to the *Factory Mode*.

The same article shares a similar phenomenon to the one described in the evolution of software across the IT Strategic Impact Grid; exploration- and utility-oriented F/OSS types are usually suited for the initiation phase, whereas a service-orientation is better for mature projects. This shows that as communities grow and mature, they tend to lower their innovation needs and instead focus on delivering stable and robust services.

De Laat (2007) proposes another way of classifying OSS projects, this time more related to socio-economic theories. The author describes two different structures: *autocratic-mechanistic structure* and *democratic-organic structure*. The former is based on autocracy, with a self-appointed leader and a high degree of vertical differentiation and centralization. Governance is meant to *control* its people, who are highly distrusted. On the contrary, the latter is based on democracy, with election processes as seen before. There is a high degree of horizontal differentiation, and low levels of centralization. Trust is an important factor here, both in the belief that members work towards the goals of the project, and that they are capable of selecting a good leader (or group). This classification mainly applies to larger communities, which de Laat (2007) defines as having several hundreds of members. For smaller projects – whether they are in an earlier stage or just within a small scope – a simpler structure suffices, with a strong self-appointed leader. This seems to be similar to the exploration-oriented project as described by Ye et al. (2005), and again indicates a shift in governance regimes as a result of evolution over time.

2.4.1 Innovation

From these theories, a model can be built to aid in describing the changes over time in OSS projects. The *innovation* dimension is named in multiple researches as a good way of differentiating between projects. Innovation is often defined as the implementation of something *novel*, although the debate on what exactly novel means is still ongoing (de Jong & Vermeulen, 2003). One way of viewing novelty is along a spectrum between *radical* and *incremental* innovation. Radical innovation is change with a significant effect on the organization, whereas incremental innovation is characterized by small, step-by-step changes. This difference is related in the way an organization approaches new technology. In the case of OSS, a project that is more stable, more *defensive*, is probably only incrementally changing the software. A project that wishes to push the technological frontier and break through the market is then more likely to radically change the software (*offensively*). The dimension of radical versus incremental innovation will be used in the theoretical model of this research. Since incremental innovation gradually changes software, as opposed to radically changing it, innovation can be measured according to the number of changes implemented at any time. Therefore, in the sense of OSS, both the average number of *commits per pull request* (which is one push to the main branch) and average number of *file changes per pull request* are indicative of the type of innovation. The following measure will be used:

$$Innovation\ Score = \sqrt{avg\ \#\ of\ commits * avg\ \#\ of\ file\ changes}$$

2.4.2 Sustainability

Another aspect of development that is often mentioned, is the *sustainability* of a project. As De Noni et al. (2013) have pointed out, one purpose of OSS governance is to secure a sufficient supply of people to sustain both development and innovation. Ye et al. (2005) also agree that sustainable development is the key to success and is partially influenced by the changing community. A well-balanced community is important, which they describe as the common 20-80 rule, in which 20% of the contributors commit 80% of all changes. In addition, they state that OSS success is directly related to the growth in the size of the community. Liao et al. (2018) studied the sustainability of OSS *ecosystems*, but it can still be used for OSS projects since in essence they are both built on communities of voluntary participation. They argue that long term contributors (similar to the core group) are significant for stable development. Furthermore, they define OS ecosystem sustainability as "the potential initiative of the software ecosystem to permanently sustain or support its own dynamic health and its evolutionary evolution". This means that the system is both in a steady development, and that it will continue to be in that state in the future. To measure sustainability, the following aspect of a community will be calculated: the *ratio between core- and peripheral developers*, which should ideally be between 20% and 50%. A core group representing less than 20% will be too small to steer the project in the right direction and thus long term success is not guaranteed. A group larger than 50% is also problematic, because then the issue is raised of having a weak inflow of new contributors to bring in novel ideas and help boost innovation. Furthermore, since they wrote more than 50% of the software code, they may feel like they are fully in control and dictate the community. The group of core contributors will be measured as the minimum number of contributors who together contribute for at least 80% of all commits in a certain time frame, as is also used in other research (Kaur & Chahal, 2022; Setia et al., 2012). The following formula will be used to measure the sustainability of an OSS project:

$$Sustainability\ score = \sqrt{core\ contributor\ ratio} * (1 - core\ contributor\ ratio)$$

The latter part of the *sustainability score* equals the peripheral contributor ratio, since both ratios combined equal to 1.

The final theoretical framework is displayed in table 1.

Table 1 Project categories based on desired project outcomes

		Innovation	
		High (more radical / offensive)	**Low (more incremental / defensive)**
Sustainability	**High**	**Example:** React by Facebook	**Example:** Android by Google
	Low	**Example:** Mobile game developed by an app developer	**Example:** Data analytics solution developed by a data scientist

2.5 Hypotheses development

Based on the literature discussed in the previous sections, some hypotheses can be built. This study is both an attempt to test if the existing theories on changing community structures hold, and to identify new factors and phenoms that cause and/or impact change. A temporal change can be measured in many different ways and across different dimensions. Furthermore, time and therefore the temporal factor is continuous, meaning that a community may change on multiple occasions across its lifespan. The following sections will each cover one of the four sub-topics as laid out in the introduction.

2.5.1 Evolutionary dynamics: going open source

The first release of OSS is a significant event and can be a determinant of a project's success. Hogarth and Turner (2005) have studied the evolution of 179 open source projects in the clinical domain using several different metrics. One of these metrics is the *evolution metric*, based on the releases of a project over time. The authors found that the longer a project dwelled in the pre-first release stage(s), the less likely it is to evolve into later stages, or never even publish its first release. Interestingly, they found that the average number of days from project announcement to first release was 128 days, or a median of 24 days. This would mean that an open source project is likely to publish its first release within the first quarter – or at least the first year –, or otherwise never publish at all and thus fail to go open source. This shows the relevance of the first release and the period before that.

Traditionally though, software is developed to the full requirements and then delivered as a "big bang" first release. In such cases, the first release is of an even greater importance since the implementation and usage fully depend on it. In OSS development however, rapid iteration is key (Hogarth & Turner, 2005) and it is not unlikely to have multiple releases within a short amount of time. This makes it easier to prioritize requirements and adapt to changes in requirements. The impact of the first release is therefore not as great compared to proprietary software. Nevertheless, since OSS development is based on the vitality of its community (Ye et al., 2005), the project will benefit of having a strong first release as it will be received better by users and thus potential new contributors, also caused by a larger exposure. This is one of the advantages of iterative development (Greer & Ruhe, 2004), leading to more support and feedback, and an increase in the size of the community (although its relative growth may level at the point when the system reaches stability (Ye et al., 2005)). More contributors will likely lead to an increase in innovation, as supported by Raymond (1999). Therefore, the first hypotheses are:

Hypothesis 1: *The growth rate of a community increases in the short term post-first release*

and thus

Hypothesis 2: *Innovation increases in the short term post-first release*

Going open source - the first release - facilitates the engagement of a diverse community of developers, who contribute to the repository by improving code quality, fixing bugs, and adding new features. This collaborative effort enhances innovation by leveraging the collective intelligence and expertise of a global community and will be measured with the average number of commits and file changes per pull request.

However, innovation does come at a cost. As discussed earlier, OSS development comes with the additional complexity of coordinating efforts of a *geographically dispersed base of volunteers* to the software (Nelson et al., 2006). With more people joining the community comes more complexity, and thus an increase in coordination efforts. Especially if a project is popular, it may attract many peripheral contributors at first that want to contribute by e.g., fixing bugs (that probably come in an abundance in the first couple of releases). This would mean that the core contributors become less meaningful in the project. Additionally, new contributors may build up the conflict in the community, and the chances of forking increases (Lerner & Tirole, 2002; Ye et al., 2005). Because of this, the sustainability decreases – at least for the period of time following the first release, until it stabilizes and/or an effective governance strategy is put in place. All this put together results in the following hypothesis:

Hypothesis 3: *Sustainability decreases in the short term post-first release*

2.5.2 Project characteristics

After identifying the changes in innovation and sustainability post-first release, it is important to reflect this evolutionary change on a larger scale, across the 2x2 matrix – the different types of OSS projects (table 1). As seen in the paper by Ye and his colleagues (2005), exploration- and utility-oriented F/OSS types (low levels of sustainability, varying innovation) are better suited for the *initiation phase*, whereas a service-orientation (high sustainability, incremental innovation) is better for mature projects. In proprietary software development, projects tend to shift from an offensive strategy to a defensive one in the long run. Therefore, the following can be hypothesized:

Hypothesis 4: *Projects that start with high, radical levels of innovation are likely to shift towards a more defensive level of innovation as they mature (shift left to right)*

Hypothesis 5: *Projects that start with low levels of sustainability are likely to shift towards higher levels of sustainability as they mature (shift lower to upper)*

If both of these hypotheses are accepted, then it would mean that a project in any category is likely to shift towards a defensive and sustainable project, similar to the service-oriented F/OSS type. However, this may not hold for all of the quadrants. As Ye et al. (2005) argued, utility-oriented type projects have a "tournament style" evolution, in which multiple solutions (similar projects) fight to win by gaining the most support. The winners may get embedded in the larger system for which they developed a solution, but the losers may stay where they are and eventually lose support. Since utility-oriented projects are started as "scratching a personal itch" (Raymond, 1999), and are usually developed by a single person or small group of people, it is similar to the fourth quadrant in the matrix. It will likely have low sustainability and low innovation levels, as it does not have a goal of "pushing the market", but merely trying to fix an issue. The following can be hypothesized:

Hypothesis 6: *The majority of projects that start with low innovation and sustainability levels are likely to stay in their quadrant over time*

Besides mapping the evolutionary shifts across the matrix, it is important to know *why* these shifts happen, or when they are more likely to occur. For example, as innovation is user-driven in OSS projects (Lerner & Tirole, 2002; Raymond, 1999), more (new) developers would likely lead to more radical innovations. According to López et al. (2017), larger communities tend to attract more people. Overall, project size is a variable that is commonly used in OSS literature to help explain or predict certain phenomena (de Laat, 2007; Sharma et al., 2012, Ye et al., 2005). Therefore:

Hypothesis 7: *Community size positively affects the number of new contributors*

Hypothesis 8: *Larger communities are more likely to increase in innovation as they grow than smaller projects*

Lastly, the governance strategy of an OSS project may have an impact on the way these communities evolve. As de Laat (2007) argues, the most important decision in OSS governance is that of who decides what code to include in the main product – also known as merging a pull request. In *centralized* governance designs, there is a small group of people (the project leader(s)) that hold this decision right. Since the relative power of this small group is large, they are less likely to accept any pull request – only from whom they trust – and therefore take on a more *protective* pull request reviewing approach (Alami et al., 2021). De Laat (2007) also motivates that OSS governance can be seen as a motivation for its members, meaning that a positive climate can increase their motivation to contribute more effort and thus increase innovation. In a centralized design, wherein members have to gain the leaders' trust in order to contribute to the project, the climate can be strict and may come off as demotivational. Therefore, the following can be hypothesized:

Hypothesis 9: *A project with more unique pull request reviewers leads to higher innovation scores over time*

In addition, if word of a positive climate spreads, this may lead to an increased inflow of new developers. One of the goals of OSS governance is to secure a sufficient supply of people (De Noni et al., 2013). Therefore:

Hypothesis 10: *A project with more unique pull request reviewers leads to a higher inflow of new developers*

2.5.3 Organizational characteristics & platform ownership

The third phase of this research dives into the effects of heterogeneous organizations supporting or founding OSS projects, and the effects of platform ownership. Based on the resource dependency theory, Ferraz and dos Santos (2022) state that OSS projects are continuously seeking resources for business survival, hence the need for a relationship with stakeholders in the first place. Projects and their communities are influenced by their environment; leaders may seek organizations that can help them with acquiring the necessary resources, ultimately leading to more external influence.

The severity of external organizational impact may differ depending on the type and intentions of the organization sponsoring the project. One aspect that may be interesting, is that of profit versus non-profit organizations. Profit organizations are interested in projects that deem commercially viable and may attempt to steer the community into a similar mindset. As seen before, some developers are motivated by their expectations of future returns and may even earn some form of income/salary (Nelson et al., 2006). Other developers are just there to support the open source movement, and do not care for income, or may even get demotivated when external (not open) organizations are getting involved (De Noni et al., 2013; Jin et al., 2007). Therefore, communities sponsored by for-profit organizations attract differently motivated people than those sponsored by non-profit organizations. A result of this could be that projects sponsored by a for-profit organization is more competitive than a non-profit organization, especially when monetary income is involved. Furthermore, for-profit sponsored communities may retain their members for a longer period of time because of the competitive and rewarding nature, compared to the more voluntary nature in non-profit sponsored projects. Therefore, the following can be hypothesized:

Hypothesis 11: *Communities sponsored by a for-profit organization have higher levels of innovation because of its competitive nature than those sponsored by a non-profit organization*

Hypothesis 12: *Communities sponsored by a for-profit organization have longer member retention than those sponsored by a non-profit organization*

Furthermore, there may be differences even between for-profit organizations. For instance, since the GitHub platform has been acquired by Microsoft in 2018 (Microsoft

News Center, 2018), the nature of innovation may change caused by this change in platform ownership. One may expect for example that Microsoft's projects have improved more in popularity and success than those founded by other for-profit organizations such as Google, as a result of the better competitive position of the multinational. Therefore:

Hypothesis 13: *The yearly number of new projects by Microsoft after their acquisition of GitHub in 2018 is higher than that of Google*

Hypothesis 14: *Microsoft's projects have performed better in innovation levels than Google after Microsoft's takeover of GitHub in 2018*

In terms of sustainability, the effect could go both ways. On the one hand, an increased popularity and therefore inflow of new contributors may cause a (temporary) decline in sustainability because of increased coordination efforts. But on the other hand, more available resources and overall ownership of the platform may allow Microsoft to deploy their best developers and communities, potentially leading to increased sustainability.

Lastly, it is expected that the effects of increased platform popularity and the availability of more resources (Lardinois, 2022) also impact those not directly touched by the improved competitive position of Microsoft. Non-profit organizations, such as the Apache Foundation, may benefit from a stronger reach on GitHub and more available features to use in their projects, without suffering under the competition of Microsoft:

Hypothesis 15: *OSS projects sponsored by non-profit organizations have increased innovation levels after Microsoft's acquisition of GitHub in 2018*

2.5.4 Contributor characteristics

In the paper by De Noni et al. (2013), the authors describe the perspective of the life cycle of an OSS project. They found that the evolution of such projects can be characterized by three different phases: invention, innovation, and diffusion. In the first phase, *invention*, the main focus is on gathering a small group of strongly motivated and resourceful developers. This group is likely to form (a part of) the core group over time. The *innovation* phase puts emphasis on attracting both leader users and firms through its governance strategy. This tends to be a much larger group than the first phase, aimed at complementing the work of the core group. In addition, most members in this group are probably not going to be long term contributors, as Lerner and Tirole (2002) found that more than 75% commit only once. As seen earlier in this paper, new-comers are more likely to abandon the project than those who joined some time ago (Kaur & Chahal, 2022). In other words, the average contributor retention rate seems to drop over time. Kaur and Chahal (2022) could only partially verify this phenomenon, since two of their five projects indicated a different abandonment pattern. Therefore, this thesis tries to verify the following hypothesis with a much larger sample size:

Hypothesis 16: *Contributors joining the project early have a higher retention rate than those joining in more mature phases*

The final phase, *diffusion*, is one that has not been included in the other phases of this research. In this phase, incentives towards outside firms are key to leverage the distribution of the software. As seen before, a foundation may be built around the community to manage their external relationships (de Laat, 2007). It is then not unlikely for such foundations to intertwine with the project and its development. This event may result in the original set of developers to leave the community because of their commitment to free information accessibility (De Noni et al., 2013). Regardless, they may deem the project as no longer technologically challenging.

Since the dataset for this research are gathered from repositories by Microsoft, Google, and Apache, it may be difficult to find this diffusion phase since most projects are already sponsored by a firm (as de Laat (2007) calls "sponsor founded" versus "community founded" projects) with its own distribution network (although there are exceptions). For example, Apache has an elected Board of Directors that does have a say in project work (de Laat, 2007). However, similar events can still take place as a result of the project's evolution. For example, it is expected that the original set of developers decreases over time, potentially up to the point where the project reaches stability and there is not much challenge left. Furthermore, this will likely result in a decreased inflow of new developers as the project evolves over time, becomes more mature and thus requires less maintenance or new features. Therefore, the following hypotheses are set:

Hypothesis 17: *The original set of core developers decreases as the project matures*

Hypothesis 18: *The inflow of new developers decreases as the project matures*

3 METHODOLOGY

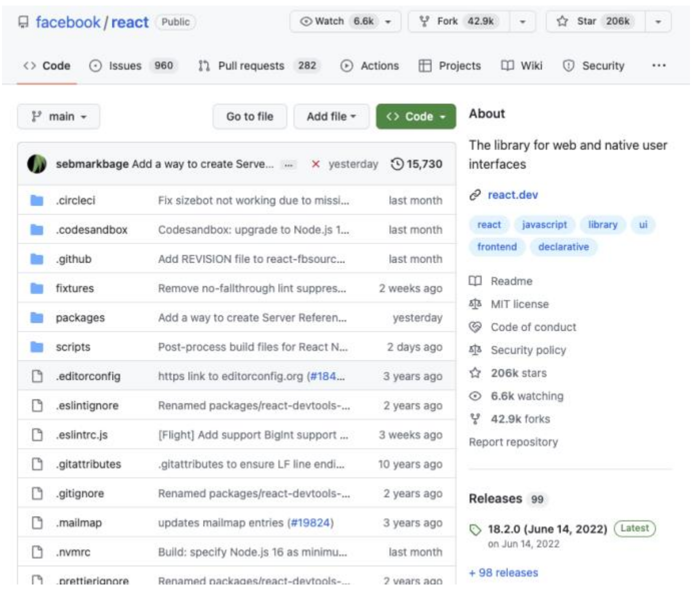
3.1 Context

Since this research is quantitative, it is important to choose a reliable, representative and fitting data source. For this purpose, GitHub has been selected as the main source of data. It is currently the most popular and biggest open source community, with over one hundred million repositories and tens of millions of users (Kashyap, 2020; Moqri et al., 2018). Among these repositories are some of the most famous pieces of OSS, such as Linux and JQuery. Therefore, data retrieved from this platform will most likely be representative of modern and active OSS communities.

GitHub has two public APIs available for both developers and researchers to extract data: GraphQL and REST (About GitHub's APIs - GitHub Docs, 2022). The differences between these two methods mainly lie with the structure and type of requests that you can make. For example, using GraphQL allows one to predetermine the structure of the data that he/she wishes to receive. In addition, multiple queries can be combined into one complex query, unlike the REST API. Since both the sustainability and innovation measures require quite a drilldown into a repository (commit- and pull request-level), being able to combine queries and therefore decrease the overall execution time is a defining benefit. Therefore, the GraphQL API will be used to extract data from publicly accessible repositories.

3.2 Data collection

As seen before, GitHub works with *repositories*, which are containers surrounding all project files and their revision history. They can be owned by an individual user, but the ownership may also be shared with others in an organization. Some communities are private, and therefore require you to have *collaborator access* in order to view and commit to that project. GitHub offers a multitude of collaboration features, such as *issue reporting* like bugs or feature requests, managing *pull requests* for incoming changes, and a discussion board for easy communication. Furthermore, a current version of the software can be put into a *release*, also managed within the repository. See figure 3 for an example repository.



sebmarkbage Add a way to create Serve... ✕ yesterday 15,730

.circleci	Fix sizebot not working due to missi...	last month
.codesandbox	Codesandbox: upgrade to Node.js 1...	last month
.github	Add REVISION file to react-fbsourc...	last month
fixtures	Remove no-fallthrough lint suppres...	2 weeks ago
packages	Add a way to create Server Referen...	yesterday
scripts	Post-process build files for React N...	2 days ago
.editorconfig	https link to editorconfig.org (#184...	3 years ago
.eslintignore	Renamed packages/react-devtools-...	2 years ago
.eslintrc.js	[Flight] Add support BigInt support ...	3 weeks ago
.gitattributes	.gitattributes to ensure LF line endi...	10 years ago
.gitignore	Renamed packages/react-devtools-...	2 years ago
.mailmap	updates mailmap entries (#19824)	3 years ago
.nvmrc	Build: specify Node.js 16 as minimu...	last month
.prettierignore	Renamed packages/react-devtools-...	2 years ago

The library for web and native user interfaces

react.dev

react javascript library ui frontend declarative

Readme
MIT license
Code of conduct
Security policy
206k stars
6.6k watching
42.9k forks
Report repository

Releases 99

18.2.0 (June 14, 2022) Latest
on Jun 14, 2022

+ 98 releases

Figure 3 React by Facebook repository on GitHub

Most of these features are represented as *objects* in the GraphQL API. It is quite simple to extract basic repository information like the creation date, default (main) branch of development, what languages are mainly written, or how many releases a project has. The following example shows how a few lines of code result in the same information as figure 3, in the structure as defined by the user:

Query	**Return**

```
{
 repository(name: "react", owner: "facebook") {
  nameWithOwner
  createdAt
  isPrivate
  isArchived
  stargazerCount
  forkCount
  defaultBranchRef {
   name
  }
  releases(first: 1) {
   totalCount
  }
 }
}
```
	```
	{
	 "data": {
	  "repository": {
	   "nameWithOwner": "facebook/react",
	   "createdAt": "2013-05-24T16:15:54Z",
	   "isPrivate": false,
	   "isArchived": false,
	   "stargazerCount": 205863,
	   "forkCount": 42919,
	   "defaultBranchRef": {
	    "name": "main"
	   },
	   "releases": {
	    "totalCount": 99
	   }
	  }
	 }
	}
	```

But when it comes to commit-level data, the queries become much more complex. This is because it involves deeper levels, which each exponentially increases the size of the result. Furthermore, GitHub has some limitations to the requests, to protect against excessive or abusive calls to their servers (Resource Limitations - GitHub Docs, n.d.).

Any object that has a deeper relationship with multiple nodes (which GitHub calls a *connection*) requires a *first* or *last* argument, which accepts values between 1 and 100. This means that for every request, only the first or last 100 results will be returned. Since most repositories have more than 100 commits (all of Google's repositories average around 900), one request is not sufficient. This is where *pagination* comes in. Pagination allows you to page through all available nodes in a connection. Every connection can include a *pageInfo* object, which results in the following data:

Return

```
"pageInfo": {
        "hasNextPage": true,
        "endCursor": "b6006201b5fdfcc5720160f169b80ddb7b8d7467 99"
}
```

hasNextPage tells the user that there are more nodes after the 100 that have just been returned. The *endCursor* can then be included in the follow-up request to retrieve the *next* 100 nodes (using the argument "after:" in the connection). This can be repeated until *hasNextPage* is false and thus all nodes have been returned. In addition to these limits, GitHub's GraphQL API also has a *rate limit* of 5,000 points per hour. Points are calculated for the complexity of the query (for example based on the number of connections and how many nodes each of these return).

For retrieving all of Google's repositories and their commits, it would take around 22,320 requests (2480 repositories * (900 average commits / 100 commits per request)). But as not all repositories are public, and some are locked or archived, the total number of repositories that can be used for analysis is lower. Unfortunately, the GraphQL API is often returning an error when too many requests are sent in a short amount of time (a timeout). This was discovered in one of the first test runs. To counter this issue, the data collection script goes to sleep for 3 seconds after every request. In most cases, this functions as expected and the queries run normally. However, the script has a built-in retry process to attempt to run the same query again after 5 more seconds when an error is returned. This results in a mostly fool-proof data collection process that can run while being idle and thus not requiring much manual intervention (but is in no way a quick process). In the cases where the retries take too long, the script is put on hold manually, to retry after some time. When all commits for a single repository have been collected, a JSON file is exported to a personal OneDrive with the name of the repository. This way, even if the script has to be stopped because of an overload or other issues, the data is still stored up to the repository that was last being processed.

The script has collected all commit and pull request data for all repositories that have at least one release (for measuring the temporal aspect) for the first 1,000 search results of Google and Microsoft. The result includes details of well over 1,000,000 commits. GitHub's search queries allow you to specify more arguments for e.g., filtering out archived and private repositories, but also limit at 1,000 results. In total, the dataset includes commit and pull request data for 895 public and open repositories, split between 418 and 477 repositories of respectively Google and Microsoft. These two organizations were chosen because they have highly varying projects and many active communities built around them. Furthermore, many have existed for quite some time and have multiple releases, enabling the temporal measurements. Lastly, commit and pull request data is collected for 890 Apache projects, to be able to research the non-profit organization hypotheses. More projects could be used than those by Google and Microsoft because the first release-requirement was not necessary here.

3.3 Research design

After the data is collected, it is time to conduct the actual research. This is done in several steps, mostly in Python, Excel, and SPSS Statistics. This section will be split according to the four phases of this research.

3.3.1 Evolutionary dynamics: going open source

This phase mainly consists of processing the early data of a project – namely pre- and post-first release. To measure the direct impact of the first release, the innovation score, sustainability score, and growth rate will be measured in the maximum of 6 months before and after the first release, where available. This time period is chosen because more than 70% of the projects in the dataset published their first release within year 0 or year 1, so a longer time period would not be useful for most projects. In addition, the measure for growth has changed to measure monthly *cumulative* growth ratios. This is to account for the months wherein no new contributors join the project, hence a growth of 0 would return, and the ratio with the next month becomes immeasurable. Furthermore, the projects that had no peripheral contributors in either the period pre- or post-first release were not considered for this phase of the research. This is due to the result being a sustainability score of 0, and because of the relatively short period of time this may not be accurate. Lastly, only projects that published their first commit at least 30 days before the first release (minimum of 1 period) were used, to exclude projects that have almost no data pre-first release. The returned dataset consists of 397 projects. The following steps were taken:

1. *Split commit and pull request data based on the first release date*

   This is the temporal aspect of this phase, and thus the data should be split based on this date per repository. GitHub's API extracts the *tag* creation date from which the release is created, instead of the release's publish date. Tagging is technically a git function and slightly different from a GitHub release. For example, a release may contain more detailed information such as changelogs and links to specific commits. A tag on the other hand is a specific point marked in a repository's history. In both cases, the repository is packaged in the state at that point in time and available to download and use by the public (if available). As tags are generally used for release versioning and exist for longer than GitHub's releases (Git - Tagging, n.d.; Olsen, 2013), this field is representative of the project's first "official" release.

2. *Calculate innovation and sustainability scores pre- and post-first release*

   Now that the data is split, both dimension measures and the cumulative growth rate can be calculated for the data before and after the first release date. This will result in the measured temporal change in the repositories across the dimensions.

3. *Conduct statistical regression*

   Next, the data will be analyzed using a paired samples T-test. With the results of this step, the hypotheses can be proven or disregarded.

3.3.2 Project characteristics

This phase of the project focuses on how and why the projects of Google and Microsoft shift across the matrix over time. Hypotheses 4, 5, and 6 will be researched as follows:

1. *Determine the projects' original quadrants based on the data of the project's first year from the first commit.*

2. *For each of these cohorts, determine the position / quadrant for every sequential year that the project has been active (at least one commit), up to a maximum of year 6. This threshold is chosen because less than 40% of the projects in the dataset are older than 6 years and that may skew the results.*

3. *For each of these cohorts, create a table based on the project's new type after every sequential year. Then, calculate the distribution of project types (in percentages) for each year, and generate a heatmap based on its values.*
4. *Additionally, perform paired samples tests between two time points to be able to statistically answer the hypotheses. The type of test is dependent on the distribution of the innovation and sustainability levels (~normal or skewed).*

Hypotheses 7, 8, 9, and 10 will be researched using a longitudinal design, similar to the study designed by López et al. (2017). The data is segmented into sections of 90 days (later also in 365 days for comparison), starting from the first commit. The association between the project's variables in period *p* and the measures in the following period *p+1* (as described in table 2) will be determined using linear regression modelling. This way, the impact of the discussed variables can be measured, and other variables may be found as additional factors. The factors are a combination of variables found in previous literature, and those used in the study by López et al. (2017).

Table 2. Description of the variables used in study phase 2

Project variables (IVs)	Measures (DVs)
Period (age)	Number of new (first-time) contributors
Community size	Growth rate (new contributors / base)
Total commits	Number of retained contributors from previous period
Total pull requests	
Total unique PR reviewers	Retention rate (retained contributors / base)
	Innovation score
	Sustainability score

3.3.3 Organizational characteristics & platform ownership

Hypotheses 11 and 12 will be researched using independent samples tests. The two samples in this case are the projects sponsored by Google and Microsoft (for-profit sponsored), and the projects sponsored by Apache (non-profit sponsored). For H12, member retention will be measured as the average number of days between a contributor's first and last commit. Some studies choose to consider a contributor *dead* after one year of no activity – a period in which the member may be "asleep" (Iaffaldano et al., 2019).

But as discussed earlier, there is no way of knowing if this contributor is ever to return to the project; and because of the knowledge loss after some time, this contributor may be considered "new" anyway (Rashid et al., 2019). Whilst the period between first and last commit is sure to contain some sleeping states, it is considered to be the simplest and most efficient measure to determine retention.

H13 will be descriptively measured using a graph displaying the yearly number of projects created for each of the organizations. H14 and H15 will be researched using independent samples tests. The data of Microsoft's, Google's, and Apache's projects will be split on the year of acquisition (2018). The tests will determine if there is a significantly different innovation score post-acquisition.

3.3.4 Contributor characteristics

This phase focuses on how the community and its contributor turnover changes over time. Hypothesis 16, 17, and 18 will be determined using a complex, descriptive design:

1. *For every unique contributor, determine their first and last commit date*

   This is needed to determine the cohort of a contributor, and to measure how long his or her retention period is.
2. *For every project, split the data into segments of 90 days (~quarters)*
3. *For each segment, determine how many contributors are new to the project based on their first commit date*

   Whenever a project has 0 new contributors in a period, it will be disregarded from the calculation for the average retention. This is because when it comes to determining retained members, a base of 0 is not useful for calculating percentages / ratios.
4. *Iterate over all existing periods after the cohort's end date and determine how many of the original contributors (in %) are still active based on their last commit date*
5. *Import the aggregated (average) data from all projects into Excel, and colour code them based on a scale of 0-100%.*

The result will be a matrix showing the average retention percentage for each period of 90 days, across the lifespan of all projects in the sample. The sample only includes projects that have been active for at least 20 quarters (5 years), to assure that there is enough history to determine retention trends. Whilst H18 can be descriptively answered using this design (but with the limitation of skipping periods with 0 new contributors), a linear regression test will also help to determine the average inflow of new developers over time (which will include zero inflow in the averages).

4 Results

All results presented in this chapter are based on correlation evidence, as causality has not been tested. Each sub-topic has its own paragraph.

4.1 Phase 1: Evolutionary dynamics, going open source

The goal of the first phase is to identify the impact of the phenom that is the first release. For each of the projects in the dataset (N=397), both the project variables and outcome variables (innovation, sustainability, and growth) have been measured based on the time period of maximum 180 days (~6 months) before and after the first release. This time period is lightly based on the study by Hogarth and Turner (2005), who state that the majority of projects in their sample published their first release within 124 days. Since this time period is relatively short - especially given the age of GitHub repositories - it has been extended to 180 days. This is also tested on the sample set for this phase. The results of this test show a mean of 21.158 periods, or approximately 635 days. However, this is not normally distributed as the median lies at just 11 periods, or 330 days. Therefore, the majority of the projects in the sample set have their first release within the first year (55.2%). This means that the maximum period of 180 days will contain data for most projects.

The descriptive statistics are displayed in table 3, paired in variables before (prefixed "*pre_*") and after (prefixed "*post_*") the project's first release.

Table 3 Descriptive statistics for H1, H2 & H3

Variable	Min	Max	Mean	Std. Deviation
pre_totalCommits	5	2285	182.86	258.810
post_totalCommits	5	1643	164.08	236.586
pre_totalPullRequests	1	982	60.53	106.046
post_totalPullRequests	1	1026	70.67	117.843
pre_averageCommitsPerPR	0	49	3.98	5.130
post_averageCommitsPerPR	0	93	4.24	6.304
pre_averageFileChangesPerPR	1	713	15.84	44.452
post_averageFileChangesPerPR	1	159	11.21	19.957
pre_innovationScore	.000	106.752	6.341	7.996

Variable	Min	Max	Mean	Std. Deviation
post_innovationScore	.000	100.683	5.881	7.321
pre_totalContributors	2	275	10.27	16.762
post_totalContributors	2	229	11.11	16.146
pre_averageGrowthRate	.00	10.00	.403	.883
post_averageGrowthRate	.00	2.237	.105	.172
pre_sustainabilityScore	.132	.385	.346	.047
post_sustainabilityScore	.152	.385	.343	.050

A paired samples T-test was used to find the correlations and impact of the variables pre- and post-first release. This test can be used to compare the means of two separate measurements taken from the same unit, in this case of two time periods. The t-test has shown that all of the above seven pairs are indeed significantly correlated ($p < 0.001$). Furthermore, the total number of commits pre-first release is significantly higher than post-first release ($t = 2.232$; $p < 0.05$), with a mean difference of 18.78. However, the number of pull requests is higher on average *after* the first release ($t = -3.161$; $p < 0.05$), but a significant difference in average commits per pull request cannot be found ($p = 0.425$). The average file changes per pull request has a lower mean post-first release ($t = 2.06$; $p < 0.05$). In addition, the community size (number of unique contributors) is higher after the first release ($p = 0.03$), although the mean difference is only 0.844. Lastly, the average growth rate is very significantly lower after the first release ($t = 7.071$; $p < 0.001$). Therefore, the null hypothesis for H1 cannot be rejected, since the measured result is of the opposite effect. This means that the growth is stronger in the months building up to the first release, and that the relative inflow of new contributors slows down after the first release.

Furthermore, the null hypotheses for H2 and H3 cannot be rejected, since the mean difference of the innovation and sustainability scores are both insignificant (p = respectively 0.304 and 0.452). The descriptive table even shows that the innovation score is slightly lower after the first release (6.341 to 5.881). The means for the sustainability scores are very close, with a mean difference of 0.002.

4.2 Phase 2: Project characteristics

4.2.1 Descriptive statistics

The second phase focuses on the temporal changes across a longer time frame, and specifically into the differences between the four types of OSS projects. To be able to measure the project type, a threshold has to be set at both of the dimensions. For this, two approaches were tested. Firstly, the mean values of the innovation and sustainability scores were used. Then, the median scores were used, followed by an evaluation between the test results. See table 4 for the descriptive statistics of 860 projects by either Google or Microsoft. This dataset is larger than that of the previous phase, because the projects with no registered first release have not been excluded here.

Table 4 Descriptive statistics for the two studied dimensions, and project age

Variable	Min	Max	Mean	Std. Deviation
innovation_score	0.000	92.331	6.044	6.366
sustainability_score	0.000	.385	0.319	0.070
full_years_active	0.000	16	4.320	2.996

The medians of the innovation and sustainability scores were respectively 4.243 and 0.340. These values show that the innovation score is distributed right-skewed, while the sustainability score is left-skewed. Specifically, more than 60% of all cases are *below* the mean innovation score. On the other hand, only around 30% is below the sustainability mean. The heatmaps were generated based on both ways of determining the thresholds. What turned out is that by using the means, the division of project types in the first year is very uneven (*min* = 25). However, by using medians, the individual subsets of data are considerably large enough (*min* = 81) to be able to imply something about each project type. This method was ultimately chosen for the final analysis, although overall results are similar. The first-year descriptive statistics and distributions (in %) for both threshold definitions are included in table 5 and 6.

Table 5 Year-on-year evolution by first-year project type, based on median thresholds

Sust. / Innov.	Year 1	Year 2	Year 3	Year 4	Year 5	Year 6	% left
Low - High	10%	10%	10%	10%	7%	6%	18.5%
High - High	31%	31%	28%	25%	23%	21%	21.8%

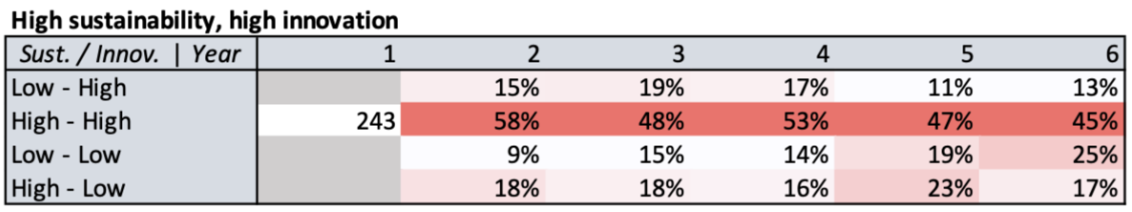
Sust. / Innov.	Year 1	Year 2	Year 3	Year 4	Year 5	Year 6	% left
Low - Low	23%	22%	24%	26%	28%	31%	42.5%
High - Low	36%	36%	39%	39%	42%	41%	35.2%

Table 6 Year-on-year evolution by first-year project type, based on mean thresholds

Sust. / Innov.	Year 1	Year 2	Year 3	Year 4	Year 5	Year 6	% left
Low - High	3%	3%	3%	3%	2%	2%	16.0%
High - High	21%	21%	19%	17%	14%	13%	19.6%
Low - Low	23%	23%	24%	27%	27%	30%	40.8%
High - Low	53%	53%	54%	54%	56%	55%	32.1%

Note that the last column in both tables, percentage left in year 6 of the projects, show clear favourites to the *incremental* (low) innovation types.

4.2.2 Shifts across the matrix



High sustainability, high innovation

Sust. / Innov. | Year	1	2	3	4	5	6
Low - High		15%	19%	17%	11%	13%
High - High	243	58%	48%	53%	47%	45%
Low - Low		9%	15%	14%	19%	25%
High - Low		18%	18%	16%	23%	17%

Figure 4 Year-on-year heatmap based on high sustainability & high innovation projects

In figure 4, one of the resulting heatmaps is given, in this instance based on year-one projects with high sustainability and innovation (*n* = 243). The other three heatmaps are attached in appendix A. The percentages are based on the distribution of the original high-high projects, for that particular year. In this case, it seems that the majority is likely to remain a high-high project in every sequential year. Some smaller portions shift across the matrix to one of the other types, which all seem to be similar in proportion for most years.

The other radical project type, low-high, has the least number of cases (*n* = 81). Nevertheless, the heatmap does seem to show a trend wherein the majority of projects move up in sustainability and turn into a high-high type project, or otherwise stay at low sustainability levels. Around a third of the projects shift right, towards incremental innovation levels, with the smallest result being the high-low group – although from year

4 onwards the dataset becomes too small to realistically compare these two types. This is because every year projects either get abandoned or are simply younger.

High sustainable and low innovative projects, the largest sample (*n* = 290), mostly stay in their quadrant or move down towards low-low projects. However, an increasing portion shifts to the left as the years go by, as the proportion staying in the high-low quadrant becomes less.

Lastly, the heatmap for low-low projects shows a similar trend in which projects either stay in their quadrant or move up towards the high sustainability type and remain incremental. But as the project matures, an increasing proportion seems to also change to radical innovation levels, just like low-high projects do. Especially the proportion shifting to the high-high type is larger from year 3 onwards.

From these results it seems that while projects commonly change sustainability levels, the majority still remains on the innovation side that they started on. The heatmaps do not clearly answer the hypotheses, but they do give an indication of trends. For example, when evaluating H4, both radical project types are more likely to stay radical (>50% dominance every year), than that they are to become incremental. However, there are certainly projects that do turn to incremental innovation, and the heatmaps simply do not indicate levels of change. It could be that even though projects are still classified as radical, they may have changed significantly compared to the first year. For H5, on average more than 50% of unsustainable projects move to sustainable levels over time. This trend seems positive towards H5, but again the levels of change are not shown.

To investigate this issue further, some paired samples tests have also been performed. Since both the innovation and sustainability scores are skewedly distributed, a Wilcoxon Signed Ranks test is appropriate (Xia, 2020). From these tests can be concluded that the innovation score lowers over time, for projects that start with radical innovation levels (> 4.243). This test has been repeated for year 1 versus year 3, year 1 versus year 6, and year 2 versus year 6. The first two results show a significant negative change in innovation levels ($t = -4.841, p < 0.001$; $t = -2.567, p = 0.010$ respectively). The differences between year 2 and year 6 are still negative, but this result is not significant ($t = -1.561, p = 0.118$). Figure 5 shows the distribution and frequency of the levels of change between year 1 and 3.

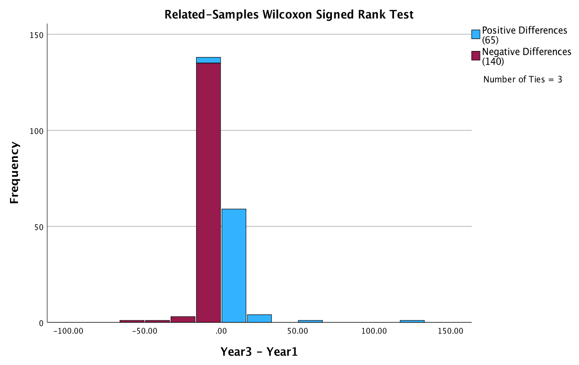


Figure 5 Innovation score difference between year 1 and year 3

To visualize these trends, figure 6 is created. It shows the trendlines for both the median and mean innovation score per year, based on projects that start with radical innovation levels. The figure shows a clear negative trend, but the averages never go below the threshold to be considered incremental. This proves the trend that the heatmaps show, wherein most projects do not shift to incremental levels over time. Therefore, H4 can only be partially proven, since the innovation levels do lower, but it not always results in a shift from left to right across the matrix.

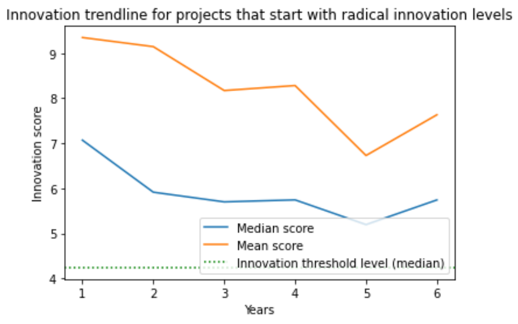


Figure 6 Innovation trendline for projects that start with radical innovation levels

From these tests can also be concluded that the sustainability score indeed rises over time, for projects that start at the unsustainable levels (< 0.34). This test has again been repeated for year 1 versus year 3, year 1 versus year 6, and year 2 versus year 6. The first two results show a significant positive change in sustainability levels ($t = 6.432$, $p < 0.001$; $t = 4.788$, $p < 0.001$ respectively). Year 2 versus year 6 still shows a positive change but is not significant at *p*-level 0.05 ($t = 0.452$, $p = 0.651$). Figure 7 shows the distribution and frequency of the levels of change between year 1 and 3. What also resulted from this test, is that the frequency of projects with a sustainability score of 0 significantly drops after year 1. For example, year one shows ~40 projects with a score of 0, but the year after this number more than halves (~18).

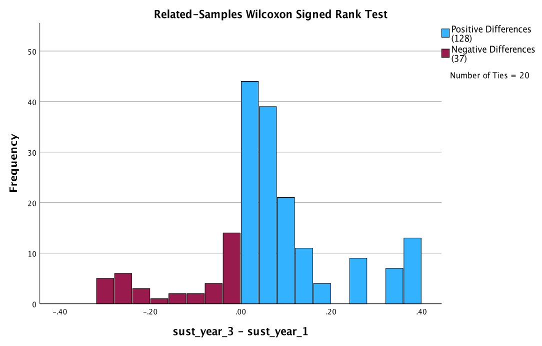


Figure 7 Sustainability score difference between year 1 and year 3

Additionally, figure 8 shows the trendline of the mean and median sustainability scores for all projects below the threshold in year 1. From this can be seen that the sustainability levels rise from the first year but drop off in terms of the rate of change (although still mostly positive). What can also be seen, is that even though the mean score is higher after the first year, it does not reach the threshold to become sustainable. The median score however does rise above the threshold from year 3 onwards. These two figures combined with the heatmaps result in accepting H5, as the sustainability scores do rise, and the majority of unsustainable projects move to sustainable levels over time.

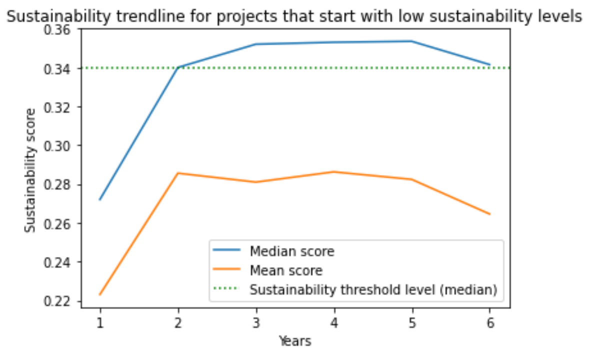


Figure 8 Sustainability trendline for projects that start with low sustainability levels

Lastly, H6 is rejected since the number of low-low projects that shift to the low-high quadrant is almost equal as those that stay in the low-low quadrant, as can be seen in the heatmap. The average percentage that remains at low sustainability and innovation levels is 37.6%, whilst 38.2% shifts to the quadrant above it.

4.2.3 The effects of project characteristics

The latter half of this phase focuses on a longitudinal study design, in which the impact of the project's characteristics in a period are measured on the DVs a period later (90 days or 365 days). Periods are only used if both the current period and the next period (p+1) have at least one commit. This is to both exclude invalid cases and periods where the next period simply "does not exist" (yet). The first test uses the number of new contributors as the dependent variable, and shows a good model fit of $R^2 = 0.723$. Community size is the most impactful variable, with a coefficient of 0.433 and a significance of $p < 0.001$. Therefore, community size is positively related to the inflow of new contributors; H7 is accepted. However, the distinct number of pull request reviewers seems to have a negative effect on the number of new contributors, with a coefficient of -0.203 at significance level $p < 0.001$. Thus, H10 cannot be accepted since the result is of the opposite effect. This indicates that having more PR reviewers actually *reduces* the inflow. Other significant variables are the period and number of pull requests, but both have coefficients close to zero. The number of commits does not have a significant effect. When adapting the model to use the growth rate as the dependent variable, the variables all show insignificant coefficients.

The next DV to model is the number of retained active contributors a period later. Results show a good model fit of $R^2 = 0.909$. All variables show a significance level of $p < 0.001$, except for the project period (age). The size of the community has a coefficient of 0.347, but this is of course a natural correlation since there can be no retained contributors without an active base. For every unique pull request reviewer, the next period is expected to retain 0.249 active contributors. The remaining two variables (commits, and number of pull requests) are both significant, but their coefficients are 0.006 and -0.002 respectively, thus not indicating a significant impact on the outcome variable. Using the retention rate as DV does show all significant variables, but none have a coefficient higher than 0.005, thus there is no good model fit ($R^2 = 0.027$).

Innovation score turns out to be a relatively hard to predict variable, with a model fit of $R^2 = 0.022$. The size of the community has a coefficient of -0.070 at $p < 0.001$, which indicates that for every increase in unique active contributors, the innovation score drops very slightly. As this effect is relatively small but still indicates a negative effect, H8 cannot be accepted. The variable that does again show a relatively high effect, is the number of unique pull request reviewers with a coefficient of 0.296 at $p < 0.001$. This confirms H9, as projects with more unique pull request reviewers lead to higher innovation scores in the next period. The other variables are all not impactful, with coefficients less than (absolutely) 0.015. Modelling the IVs on the sustainability score as DV shows no significant results.

Afterwards, the tests were repeated for periods of 365 days (~1 year). The results and effect sizes are mostly the same, but some new effects were found in the results as well. For example, it turns out that the older the project, the less likely it is to attract new contributors. With a coefficient of -0.445 at $p < 0.001$, H18 can be accepted already as the (nominal) inflow of new contributors decreases as projects age. The effect of community size on this inflow is also larger compared to the previous results (0.622 versus 0.433). A project's age in years also seems to have a negative effect on the number of retained contributors, but only slightly as the coefficient is -0.064 at $p < 0.001$. The effect of the unique number of pull request reviewers on retention is lower compared to quarterly periods (90 days), with a coefficient of 0.124.

4.3 Phase 3: Organizational characteristics & platform ownership

4.3.1 For-profit vs. non-profit sponsored

The third phase tackles the differences between OSS projects sponsored by for-profit and non-profit organizations. In the dataset, Google and Microsoft are the for-profit organizations, while the Apache Software Foundation is the non-profit organization. The for-profit organizations have 859 projects in this subset, spanning between 2001 and 2023. The early years did not have many spawns of OSS projects, as the real growth started around 2011. But as this is three years after the launch of GitHub in 2008, this is no surprise. The projects that started before 2008 were simply migrated from another platform to GitHub, which then keeps the original history of commits. The Apache Foundation did start more OSS projects earlier than Google or Microsoft, as the non-profit foundation has 882 projects spanning between 1998 and 2023. The subset even contains data for 152 projects, or 17%, that migrated to GitHub at a later point in time. This difference holds some natural conclusions, such as a higher retention duration (in days) since there are simply more and older projects. Some descriptive statistics are included in table 7.

Table 7 Descriptive statistics of for-profit versus non-profit sponsored OSS projects

Organization type	N	Avg. Innovation Score	Avg. Sustainability Score	Avg retention (days)
For-profit	**859**	**6.039**	**0.319**	**210.687**
Google	*392*	*4.197*	*0.318*	*246.060*
Microsoft	*467*	*7.585*	*0.320*	*180.995*
Non-profit	**882**	**6.096**	**0.323**	**365.867**

What can immediately be seen, are the differences between Google and Microsoft as both for-profit organizations. While the innovation is much more radical at Microsoft than Google, the contributor turnover seems to be significantly higher as a result of a shorter retention period. Nevertheless, the non-profit organization has higher scores on all three variables when compared to the total for-profit subset.

To verify these results statistically, a Wilcoxon rank-sum test was executed to compare the independent samples. Note that, for the sake of leaving out the skewed distribution of yearly projects between the samples, the subset was cut to only use projects created (or first committed to) from 2014 and before 2023. The first result is that the innovation scores are not significantly different from each other (t = 1.367; p = 0.172). Therefore, H11 cannot be accepted. In addition, the sustainability scores are significantly larger for non-profit sponsored projects than the for-profit sponsored projects (t = 3.095; p = 0.002). For retention, ultimately two measures were used to determine the turnover in a project. The first measure calculates the average number of days between the first and last commit of any one contributor. But since this measure naturally results in lower numbers as the projects are published closer to 2023 (as of writing), the second measure calculates the average retained members on a yearly basis from the year before. What turned out is that both measures favour the non-profit sponsored projects. The average retention in days differs with t = 2.787 at p = 0.005, and the average retained percentage differs t = 3.485 at p < 0.001. Therefore, non-profit sponsored projects show both higher retention periods *and* retention rates compared to for-profit sponsored projects; H12 is rejected.

4.3.2 The effects of Microsoft's acquisition of GitHub

The next part of this phase attempts to research the results of the 2018 GitHub platform acquisition by Microsoft. The first expected impact is that the number of projects that Microsoft publish on GitHub increases post-2018. Furthermore, as H13 hypothesizes, the effect of this acquisition is expected to be higher than the effect on Google as the competitor. Figure 9 shows the yearly number of projects created for each of the three organizations. This figure is visualized based on all public repositories per year, that are not locked, have at least one commit to the main branch, based on their creation date. At first, this was attempted based on the first commit date, but the GraphQL API does not allow ordering by commit date. The consequence of this is that the figure does not take into account a merged repository's history, but this effect should be quite minimal as the years draw further away from GitHub's release in 2008. Note that the year 2023 is not complete, as this data has been retrieved in June.

The figure clearly shows that the yearly number of projects created by Microsoft are significantly larger from 2015 onwards. After 2018, this number increases for two more years, and then stays stable in the following years. For both Google and Apache, this is not the case: Google's numbers remain stable around the 200 mark, whilst Apache seems to be reversing their positive trend from 2018 and further. Therefore, H13 can be considered as true, but the results simply show that both organizations continue their previous trends. Only Apache fails to continue in their trend of more new projects created yearly (although they had a spike in 2017 of over 500 projects created).

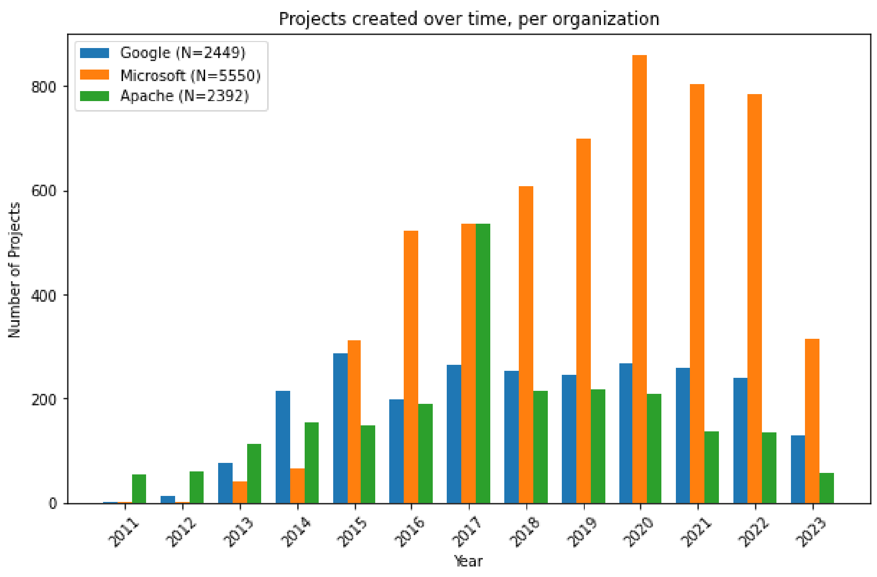


Figure 9 Projects created over time, per organization

In addition, it is expected that the innovation scores are also affected by Microsoft's acquisition. Two independent samples tests were held (Wilcoxon rank-sum test). The results show that there is no significant difference in the innovation scores between projects created before 2018, and 2018 and beyond ($t = 0.219$; $p = 0.826$). Interestingly, figure 10 shows that Microsoft's radical innovation trend is reversing after a delay of one year post-acquisition. For Google, the results show that the innovation scores have decreased after 2018, but this cannot be proven at p = 0.05 (t = -1.522; p = 0.128). Therefore, H14 is rejected.

The same test has been executed for Apache's projects. While the results indicate a decrease in innovation scores from 2018, this cannot be confirmed significantly (t = -0.702; p = 0.483).

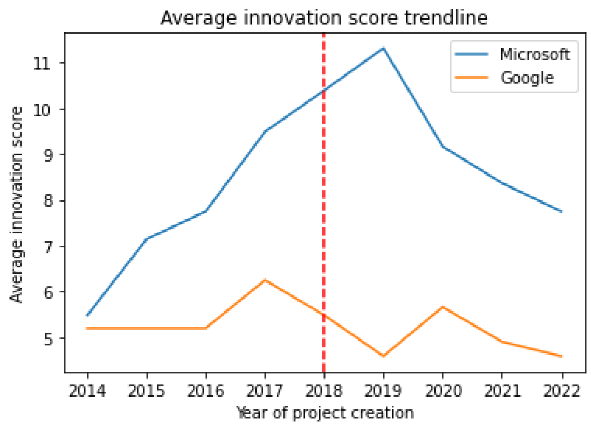
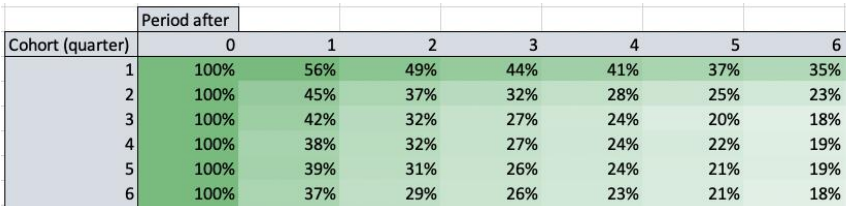


Figure 10 Average innovation score trendline, for Microsoft and Google

4.4 Phase 4: Contributor characteristics

4.4.1 Retention

The last phase looks at the turnover of the community in the long term. Retention, or the retained number of contributors over time, is the first topic. As discussed before, this is based on the first and last commit per contributor. Using 425 projects of Microsoft and Google that have at least ~5 years of commit history (20 periods of 90 days), a retention heatmap has been created. The full retention table is included in appendix B, but a section of it can be seen below in figure 11. The table shows that the contributors in the first cohort are most likely to keep contributing in the periods to follow. This is probably the core contributor group, since they commonly consist of the developers who initiated the project and are thus included in the first cohort. What can also be seen, is that the retention ratios lower as people are joining in later cohorts. For example, only 21% of developers that join in the tenth cohort are still active after a year since their first commit, whilst the second cohort shows a retention rate of 28%. This is especially visible in figure 12, which shows the average retention ratios per year of joining the project. Based on these results, H16 can be confirmed as contributors joining the project early do have a higher retention rate than those joining in more mature phases. Especially the differences between the first and second year are significant. Furthermore, H17 can also be confirmed, as the original set of core developers (those in the first cohort) decreases over time, with only around 25% remaining after 3 years.



	Period after						
Cohort (quarter)	0	1	2	3	4	5	6
1	100%	56%	49%	44%	41%	37%	35%
2	100%	45%	37%	32%	28%	25%	23%
3	100%	42%	32%	27%	24%	20%	18%
4	100%	38%	32%	27%	24%	22%	19%
5	100%	39%	31%	26%	24%	21%	19%
6	100%	37%	29%	26%	23%	21%	18%

Figure 11 Subset of the average retention per cohort (period of 90 days)

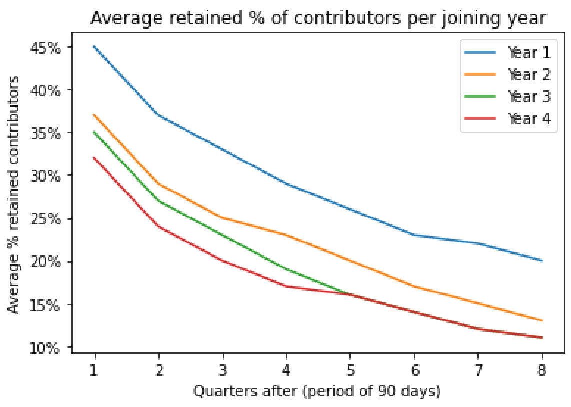


Figure 12 Average retention ratio per year of joining

4.4.2 Inflow of new contributors

The second topic of this phase is on the inflow of new contributors over time. This was measured based on the same data as was used for the retention table previously, but now also including the cohorts that had 0 inflow. For every cohort, the mean inflow is calculated and ran through a linear regression test. The results show a model fit of $R^2 = 0.26$, and indeed shows a declining trend over time, albeit the slope is only -0.061 ($p = 0.022$). Figure 13 shows the calculated linear model, and while the inflow significantly drops after the first cohort, the years after look rather stable until around year 3 when it starts dropping slightly. Nevertheless, H18 can be confirmed.

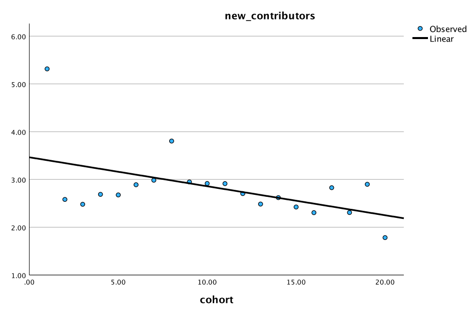


Figure 13 Inflow of new contributors over time, per cohort of 90 days

4.5 Hypotheses overview

#	Hypothesis	Result
H1	The growth rate of a community increases in the phase post-first release	🟥
H2	Innovation increases in the short term post-first release	🟥
H3	Sustainability decreases in the short term post-first release	🟥
H4	Projects that start with high, radical levels of innovation are likely to shift towards a more defensive level of innovation as they mature (shift left to right)	🟩
H5	Projects that start with low levels of sustainability are likely to shift towards higher levels of sustainability as they mature (shift lower to upper)	🟩
H6	The majority of projects that start with low innovation and sustainability levels are likely to stay in their quadrant over time	🟥
H7	Community size positively affects the number of new contributors	🟩
H8	Larger communities are more likely to increase in innovation as they grow than smaller projects	🟥
H9	A project with more unique pull request reviewers leads to higher innovation scores over time	🟩
H10	A project with more unique pull request reviewers leads to a higher inflow of new developers	🟥
H11	Communities sponsored by a for-profit organization have higher levels of innovation because of its competitive nature than those sponsored by a non-profit organization	🟥
H12	Communities sponsored by a for-profit organization have longer member retention than those sponsored by a non-profit organization	🟥
H13	The yearly number of new projects by Microsoft after their acquisition of GitHub in 2018 is higher than that of Google	🟩
H14	Microsoft's projects have performed better in innovation levels than Google after Microsoft's takeover of GitHub in 2018	🟥
H15	OSS projects sponsored by non-profit organizations have increased innovation levels after Microsoft's acquisition of GitHub in 2018	🟥
H16	Contributors joining the project early have a higher retention rate than those joining in more mature phases	🟩
H17	The original set of core developers decreases as the project matures	🟩
H18	The inflow of new developers decreases as the project matures	🟩

5 DISCUSSION

5.1 Context

The open source software (OSS) movement has seen a rapid increase in popularity over the past decade, with many popular platforms being built on the basis of OSS. This type of development comes with added complexities as a result of its voluntary and geographically dispersed community. Many researchers have attempted to study these complexities and how communities or foundations can cope with them. Most of the studies on OSS governance are focused on identifying the elements and how it differentiates from traditional development governance, whilst some have also identified that these mechanisms are in fact dynamic and may evolve over time. Such evolution is argued to be caused by both changes in the community (its contributing members), and in the OSS system on itself. How these systems and communities evolve seems to depend on the goal of the system and the structure of the community. With several researchers mentioning it as suggestions for further research, and the fact that most of the found studies are qualitative, this thesis aims to identify evolutionary patterns in OSS projects through a quantitative study of contributor commit data. The goal is to then relate the existing theory on the identified patterns, and attempt to describe what implications these evolutions have on the OSS project and its community.

From the literature, two major dimensions of an OSS project have been identified: *innovation* and *sustainability*. Innovation refers to the innovative nature of a project in terms of how radically or incrementally new changes are made. It is measured based on the average number of commits and file changes in a *pull request* (PR) – a change to the main directory of the system. Where the innovation score is based on the project, the sustainability score is more based on the underlying community. It is measured using the core contributor ratio, which shows how well supported the community is by those that belong to the inner circle of the community as a result of their commitment to the project. Additional measures found in the literature are the *growth rate*, i.e., how fast the community grows, unique *pull request reviewers*, and *retention rates*.

5.2 Evolutionary dynamics: going open source

The first phase of the research was aimed at measuring the impact of an OSS project's first release. It was expected that the growth rate of a community would increase post-first release, and with that, result in an increased level of innovation caused by more contributors as identified by Raymond (1999). The results show that the community indeed grows in size in the period after the first release, and the community merges more pull requests than before. But as the number of changes per pull request do not change, the innovation score does not increase, contrary to the hypothesis. In addition, both the number of commits and number of file changes is significantly higher pre-first release. This implies that while there are *more contributors* working on the project and merging *more pull requests*, they produce *less changes* than before. This could be explained by what Raymond (1999) describes as Linus' Law: *given enough eyeballs, all bugs are shallow* - meaning that more users (and co-developers) lead to both more identified bugs and quicker bug fixes. Bug fixes are usually small fixes, which may be what is happening here as the average number of file changes per PR becomes significantly less after the first release. In general, this indicates that the innovation becomes less radical and more aimed at maintenance, but the results cannot significantly show a change in innovation scores.

Furthermore, while the community does grow, the *rate* at which it does is significantly less after the first release. This is an interesting result as it seems counterintuitive of the "going open source" phenomenon, which would be expected to attract new contributors quicker than before. However, this could be because of the fact that the projects are sponsored by organizations and thus do not always follow a true voluntary, open source style of development and community building (at least not at the start of the project). Members may have been assigned to the project by the organization. Or the project may have been private at the time of the first release, blocking external developers to contribute to the project.

The last result of this phase shows that the sustainability score does not significantly change. This implies that the core contributor ratio is rather stable, at least for the selected time period of 1 year (6 months pre-first release versus 6 months post). Thus, as the community grows in size, so do the number of core contributors. The following patterns have been identified with regards to the first release:

- Communities still grow post-first release, but slower than before
- Pull requests include more small changes than before, such as bug fixes
- The core contributors grow at a similar rate to that of the peripheral contributors

5.3 Project characteristics

In the second phase, the heterogeneity between projects was researched, based on the theoretical two-by-two matrix across the innovation and sustainability dimensions. Previous studies have shown that development projects tend to start with radical levels of innovation, with the goal of beating the market or “pushing the frontiers of software development” (Ye et al., 2005). But as these projects mature, they are likely to shift towards more defensive strategies (Nolan & McFarlan, 2005). The results show that innovation scores do go down for projects that start with radical innovation levels, thus becoming more defensive over time. However, the results also show that these types of projects do not necessarily move to the other side of the framework, as the threshold is often not reached. Perhaps this phenomenon is more likely to be found in even more mature stages, such as projects existing for 10 years or longer, but there are currently not enough projects with this age to confirm or deny the theory. An explanation of these small changes in innovation over time could be that the rapid iteration style of development is quite stable. If OSS projects work with project management methodologies such as Agile Scrum, they are likely to know exactly the amount of work that is viable per sprint. A sprint could result in a (merged) pull request, and thus meaning that every pull request has a relatively similar number of commits and file changes (although the number of features to be implemented will drop naturally over time).

The paper by Ye and his colleagues (2005) also states that mature projects are likely to have higher levels of sustainability than before. The results show that this is indeed true, at least for projects that start with low levels of sustainability. Especially the first year of a project shows levels far below the threshold to be considered sustainable for this study. This implies that projects in their first year are likely to have a small set of core contributors, or that the majority of contributors make only small contributions to the project (such as identified by Lerner and Tirole (2002)). As explained before, this could be as a result of Linus’ Law, as more bugs are identified as the user base grows. Especially at the start of any project, there could be much attraction to potential contributors caused by a higher expectation of bugs (called “kinderziektes” in Dutch, meaning child’s diseases because of its young age). What can also explain these lower levels, is the fact that first year projects commonly have a sustainability score of 0. A score of 0 means that all contributors are core contributors (they together made 100% of all commits). This usually happens when there are only a handful of contributors. As the years go on, these communities increase in size, more peripheral contributors join in, and the sustainability score goes up. Overall, the majority of unsustainable projects move to a sustainable level over time.

Another interesting finding is that the percentage of projects remaining after 6 years of development is the largest for the first-year *incremental* innovation projects. OSS projects that start with high, radical innovation levels have only around 20% left, whilst incremental projects average around 40%. This could indicate that both Microsoft and Google have a preference for incremental approaches, since it seems to work best for maintaining longer-term projects. However, this obviously depends on the relative age of the project compared to the current year. Perhaps both organizations used to apply an incremental approach years ago, and that they may now be starting more projects with radical approaches. This can be an interesting topic for another study.

The next task was to identify the reasons *why* projects shift across the matrix as they mature. The results show that larger projects (in terms of members) attract more new contributors, confirming the statement by López et al. on large neighbourhood communities attracting more people to their forum (2017). This specific result in the case of OSS projects can be explained with the study of Kaur and Chahal (2022). They argue that having a large group of peripheral developers are like the "pool" from where core members may ascend, and ultimately help the project to grow. What was also expected, is that large projects are more likely to increase in innovation as a result of their increased attraction. The results however do not confirm this, and it even shows a slight *decrease* of innovation for larger communities. This may again be explained by the fact that most contributors only commit once to an OSS project (Lerner & Tirole, 2002), and large communities may attract and contain a larger number of such type of contributors. Lastly, the unique number of pull request reviewers turns out to be an impactful project variable. The more unique PR reviewers there are, the *less* new contributors are attracted to the project. This is contrary to the expectation based on the study by Alami and colleagues (2021), which states that a more transparent PR reviewing approach (with many reviewers) is more desirable. But at the same time, more unique PR reviewers also leads to *increased* retention of existing contributors. This would imply that project climate (in terms of PR reviewing) as part of the governance strategy is more effective for retaining *existing* contributors than it is for attracting *new* contributors. When comparing this to the study by De Noni et al. (2013), this apparently is not effective towards the goal of OSS governance to secure a sufficient supply of people. A limitation of this finding is that the retention *rate,* i.e., the percentage of contributors that is still active in the next period, is not quite correlated with any of the studied variables. The unique number of PR reviewers does have a positive effect on the innovation levels in the following period. This is in line with the study by de Laat (2007) which states that OSS governance can be seen as a motivation for its members, meaning that a positive climate can increase their motivation to contribute more effort – and thus, innovation. In total, the following patterns have been identified in the second phase:

- OSS projects that start with high levels of innovation shift to a more defensive approach over time, although most still stay on the radical side
- OSS projects that start with low levels of innovation usually stay in the incremental levels over time
- OSS projects that start with low levels of sustainability are especially low in their first year, but increase to sustainable levels from the second year and further
- The size of the community is a good attraction factor towards new contributors
- A positive project climate (in terms of the pull request reviewing process) is more effective for retaining contributors than it is to attract new contributors
- A positive project climate (in terms of the pull request reviewing process) has a significant positive effect on the innovation scores in the next period

5.4 Organizational characteristics & platform ownership

The third phase was focused on finding the heterogeneity across organizations, specifically between for-profit competitors (Google and Microsoft), and between for-profit and non-profit organizations (Apache). OSS projects are continuously seeking resources for survival and may initiate a relationship with a formal organization (if not already founded by one). This is commonly called an organizationally sponsored OSS project (De Noni et al., 2013). Although these projects mainly seek resources, they usually have to cope with the fact that there is now an external influence on the project and its community (De Noni et al., 2013; Ferraz & dos Santos, 2022). The extent of this influence depends on the type and intentions of the organization sponsoring the project.

Firstly, the comparison between for-profit and non-profit sponsored projects shows clear differences. Apache (non-profit) has been active in the open source scene for much longer than Microsoft and Google have. This directly shows in the finding that the average retention is much higher for Apache, where each contributor is active for an average of one year (365 days). A different test also showed that the retention *rates*, i.e., the percentage of contributors remaining in the following year, are also higher at Apache's projects. In addition, the sustainability score is significantly higher. No significant difference could be found with regards to the innovation scores, although Apache's scores are higher than Google's specifically. Overall, these findings imply that contributors prefer contributing to non-profit sponsored projects compared to for-profit sponsored projects as they are more likely to remain active in the community. This complements the studies by de Laat (2007) and De Noni et al. (2013), which explain that participants of the community may leave the community because of their commitment to free information accessibility. Apache is the purest open source, voluntary form of development compared to Google and Microsoft, and these results indicate that people are more interested in belonging to open, non-profit types of communities. This may be what Markus (2007) describes as "open" versus "gated" communities. Furthermore, this is also in line with Jin et al. (2017), who argue that commercial interests taint the "gift culture" characterizing OSS communities. Apparently, the expectations of future returns or other motivations such as a possible position at Google or Microsoft do not have significantly as much of an impact on their position towards the community or project. As Markus (2007) affirms, a positive climate in an OSS project, characterized by democratic governance, holds greater efficacy in motivating contributions compared to formal rewards and private benefits. Another explanation could be that the for-profit projects are too focused on efficiency or standardization, and not on something "new and interesting" (De Noni et al., 2013).

Next, the results also show clear differences between both for-profit organizations. For example, Microsoft's projects are both more radical in terms of innovation, and slightly more sustainable than Google's. However, the average retention is much higher at Google, meaning that an average contributor is active for a longer period than those active in one of Microsoft's communities. Furthermore, Microsoft is much more active on GitHub than Google is, with more than double the number of projects created. The event of Microsoft's acquisition of GitHub in 2018 was argued to be impactful on the researched organizations, but no significant differences could be found in the data. The only shift was actually at Microsoft's projects, as the average innovation score decreased for projects created after 2019 (but are still at relatively radical levels). In total, the following patterns were identified with regards to the heterogeneity across organizations:

- Contributors value an open and voluntary climate more than formal rewards and private benefits, and therefore choose to participate longer in non-profit sponsored OSS projects than for-profit sponsored projects
- There are significant differences in development approaches between for-profit organizations
- Microsoft is much more active on GitHub than Google
- Microsoft's acquisition of GitHub has no significant influence on other organizations active on the platform

5.5 Contributor characteristics

The last phase specifically focuses on the turnover (acquisition and retention) of a community in the long term. The results of this phase conclude that contributors joining the project early have a higher retention rate than those joining later. This result confirms the partially verified outcome of the paper by Kaur and Chahal (2022). Another finding is that the falloff after the first period is substantial. Between 50- to 70% of contributors are not active anymore the quarter after their first commit. This pattern can be clarified by the finding of Lerner and Tirole (2002), who concluded that more than 75% of contributors quit after their first commit. Therefore, this shows the importance of retention methods such as onboarding processes, contributor motivation, and a responsive community (Lerner & Tirole, 2002). The results have also shown that the original set of developers (in the first quarter cohort) are the largest group of new contributors on average. This group does decrease in size over time, but at the slowest rate compared to all following quarters. The implication for this is that the original developers have the strongest sense of loyalty to the community. The last result of this phase shows that the inflow of new contributors decreases over time, or in other words, the age of the project is negatively related to the inflow. This can be caused by less need for maintenance, or there may simply be fewer challenges left as the number of (potential) features decreases with each release. However, a large portion of this decrease can be explained by the fact that the first cohort has a much higher inflow than later cohorts. All in all, the following patterns have been identified:

- Early joiners to the project have a longer retention period than late joiners

- The first quarter of the project has both the highest inflow of contributors, and has the strongest retention rates over time
- The biggest falloff occurs in the period after a developer's first commit, highlighting the importance of retention methods
- As an OSS project ages, the inflow of new contributors decreases

5.6 Academic and practical relevance, limitations & recommendations

This research adds to the literature by laying out the temporal evolutions in OSS projects. It differentiates itself from existing research by using quantitative models to both confirm or deny previously found phenoms and identifying new patterns. Furthermore, the categorization model across the dimensions *innovation* and *sustainability* is new and could prove to be useful for other studies as well. For OSS developers and core members alike, the identified patterns can help to both predict events and aid in the preparation of such events.

However, the research does hold some limitations. For example, the datasets include only projects that were sponsored by either Google, Microsoft, or Apache. It has been found that there are significant differences between sponsor-founded and community-founded projects. Therefore, the results of this study cannot be generalized towards *community-founded* projects without additional research. Additionally, the results between Google and Microsoft have shown to be quite diverse, meaning that it does not guarantee that the same results will hold for other for-profit organizations as well. Also, only one non-profit organization was included (Apache) and is therefore not representative of all non-profit-sponsored OSS projects. Another limitation of the study is the fact that a contributor is only deemed "active" if he or she *committed* to the project's main branch. But as identified in some other studies, there are many ways in which someone can contribute to a project besides committing, such as placing forum posts or reviewing pull requests. It could be an interesting research approach if a similar study is done using all types of contributions to a project. Another downside to basing activity purely on commits, is the fact that many OSS projects have *bots* (such as "slo-generator-bot", "dependabot"[1], or "renovate-bot"[2]). These are not real contributors and may impact the retention rates slightly as these bots can be active through the entire lifespan of a project. Also, as a user

[1] https://github.blog/2020-06-01-keep-all-your-packages-up-to-date-with-dependabot/
2 https://github.com/renovatebot/renovate

can create multiple accounts on GitHub, the number of unique contributors per project may be slightly too high. Lastly, not all pull request reviewers may be included in the results, since only merged PRs are included. However, this is expected to be a relatively small difference as most reviewers have probably merged at least one request.

Further research could focus on similar topics based on *community-founded* projects. It would be interesting to quantify the differences between those and sponsor-founded projects. Furthermore, future research could try to identify the relationship between retention (or contributor turnover) and innovation and sustainability. As identified in the literature, high levels of turnover lead to fresh energy and ideas. The effect of this on innovation levels, combined with the limitation of Brooks' Law, remains to be identified. Additionally, high levels of turnover may hold a negative effect on the stability or sustainability of a project. Another interesting approach could be to research the specific characteristics of core contributors, such as their turnover, and at what points in the project's lifespan contributors are more likely to join or leave the group. Finally, future research could identify the best approaches for *long term success* in OSS based on innovation approaches and contributor governance designs. This study merely identified common patterns in OSS projects, but no measure of "success" was used to classify the outcomes of such patterns.

6 CONCLUSIONS

The purpose of this thesis is to answer the following research question:

> What evolutionary patterns can be identified with regards to the community and its approach to innovation and sustainability of open source software development projects?

A quantitative research has been carried out based on the data of over 1,500 open source software projects created by Google, Microsoft, or Apache. This data has been retrieved through the GraphQL API of GitHub – the world's largest open source development platform.

Based on the academic literature, a framework has been developed to help categorize open source projects. The trade-off between *innovation* and *sustainability* in such projects are the pillars of this framework.

The results show that the first release of an open source project impacts the development processes of the community, in that more contributors join the project and the attention to smaller features and fixes is higher than the phase up to this moment. Innovation and sustainability are not particularly impacted, though.

The results do show that projects change in their innovation and sustainability levels in the long term. Projects with high (offensive) levels of innovation tend to shift to a more defensive approach over time, but mostly stay relatively radical. Projects with low levels of sustainability are likely to increase above the sustainable level after the first year. The best predictor for inflow of new contributors turned out to be the size of the community, whilst having more pull request reviewers is more effective for *retaining* contributors rather than *attracting* them. The latter is also a good predictor for the level of innovation.

In addition, contributors have a much longer retention period when working on non-profit sponsored projects than on for-profit sponsored projects. This seems to imply that they value an open and voluntary climate more than formal (future) rewards and private benefits. The results also show that there are significant differences even between for-profit organizations, and therefore do not always imply a better performance on innovation or sustainability. Lastly, a change in platform ownership did not have a significant influence on other organizations active on the platform.

The last results show that contributors joining a project early in its lifetime have a longer retention period than those joining in later phases. Furthermore, the first quarter of a project has the strongest inflow and retention compared to all following periods. The biggest falloff in any period occurs right after a contributor's first commit. Finally, the inflow of new contributors decreases as open source projects age.

These are the identified evolutionary patterns in open source software projects and show that while there are inherent differences between such projects, they do commonly follow the same or similar events. The degree of change is dependent on organizational and project characteristics. As this research is focused on solely sponsored open source projects, further research could focus on examining community-founded projects and comparing them with the results of this study. Additionally, investigating the relationship between contributor turnover, innovation, and project sustainability would be valuable. Moreover, exploring the characteristics of core contributors in open source projects, such as turnover, presents a promising direction for future research. Lastly, delving into effective approaches for long term success in open source software development, considering innovation strategies and contributor governance designs, could provide valuable practical guidance for project maintainers and contributors alike.

7 References

About GitHub’s APIs - GitHub Docs. (2022, November 28). GitHub Docs. https://docs.github.com/en/rest/overview/about-githubs-apis?apiVersion=2022-11-28

Alami, A., Pardo, R., Cohn, M. L., & Wasowski, A. (2021). Pull Request Governance in Open Source Communities. IEEE Transactions on Software Engineering, 48(12), 4838–4856. https://doi.org/10.1109/tse.2021.3128356

AlMarzouq, M., AlZaidan, A., & Dallal, J. A. (2022). The Relevance of SourceForge Data in the Age of GitHub. ACM Sigmis Database. https://doi.org/10.1145/3571823.3571830

Brooks, F. P., & Brooks, F. P., Jr. (1975). The Mythical Man-month: Essays on Software Engineering. Reading, Mass. ; Don Mills, Ont. : Addison-Wesley Publishing Company.

De Jong, J. P., & Vermeulen, P. (2003). Organizing successful new service development: a literature review. Management Decision, 41(9), 844–858. https://doi.org/10.1108/00251740310491706

Di Tullio, D., & Staples, D. S. (2013). The Governance and Control of Open Source Software Projects. Journal of Management Information Systems, 30(3), 49–80. https://doi.org/10.2753/mis0742-1222300303

Eseryel, U. Y., Wei, K., & Crowston, K. (2020). Decision-making Processes in Community-based Free/Libre Open Source Software-development Teams with Internal Governance: An Extension to Decision-making Theory. Communications of the Association for Information Systems, 484–510. https://doi.org/10.17705/1cais.04620

Feller, J., & Fitzgerald, B. (2002). Understanding open source software development. Addison-Wesley Longman Publishing Co., Inc. eBooks. http://ci.nii.ac.jp/ncid/BA56049846

Feller, J., Fitzgerald, B., Hissam, S. A., & Huff, K. R. (2007). Nonprofit Foundations and Their Role in Community-Firm Software Collaboration. The MIT Press eBooks. https://doi.org/10.7551/mitpress/5326.003.0028

Git - Tagging. (n.d.). https://git-scm.com/book/en/v2/Git-Basics-Tagging

Greer, D., & Ruhe, G. (2004). Software release planning: an evolutionary and iterative approach. Information & Software Technology, 46(4), 243–253. https://doi.org/10.1016/j.infsof.2003.07.002

Iaffaldano, G., Steinmacher, I., Calefato, F., Gerosa, M., & Lanubile, F. (2019). Why do developers take breaks from contributing to OSS projects? A preliminary analysis. arXiv (Cornell University). https://doi.org/10.1109/soheal.2019.00009

Kashyap, N. (2020, March 4). GitHub's Path to 128M Public Repositories - Towards Data Science. Medium. https://towardsdatascience.com/githubs-path-to-128m-public-repositories-f6f656ab56b1

Kaur, R., & Chahal, K. K. (2022). Exploring factors affecting developer abandonment of open source software projects. Journal of Software, 34(9). https://doi.org/10.1002/smr.2484

Lardinois, F. (2022, October 26). Four years after being acquired by Microsoft, GitHub keeps doing its thing. TechCrunch. Retrieved June 14, 2023, from https://techcrunch.com/2022/10/26/four-years-after-being-acquired-by-microsoft-github-keeps-doing-its-thing/

Lerner, J., & Tirole, J. (2003). Some Simple Economics of Open Source. The Journal of Industrial Economics, 50(2), 197–234. https://doi.org/10.1111/1467-6451.00174

López, C., Farzan, R., & Lin, Y. (2017). Behind the Myths of Citizen Participation. ACM Transactions on Internet Technology, 18(1), 1–28. https://doi.org/10.1145/3093892

Merging a pull request - GitHub Docs. (n.d.). GitHub Docs. https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/incorporating-changes-from-a-pull-request/merging-a-pull-request

Microsoft News Center. (2018, June 4). Microsoft to acquire GitHub for $7.5 billion - Stories. Stories. https://news.microsoft.com/2018/06/04/microsoft-to-acquire-github-for-7-5-billion/

Moqri, M., Mei, X., Qiu, L., & Bandyopadhyay, S. (2018). Effect of "Following" on Contributions to Open Source Communities. Journal of Management Information Systems, 35(4), 1188–1217. https://doi.org/10.1080/07421222.2018.1523605

Nelson, M., Sen, R., & Subramaniam, C. (2006). Understanding Open Source Software: A Research Classification Framework. Communications of the Association for Information Systems, 17. https://doi.org/10.17705/1cais.01712

Nolan, R. L., & McFarlan, F. W. (2005). Information technology and the board of directors. Harvard Business Review, 83(10), 96–106, 157.

Olson, R. (2013). Release Your Software. The GitHub Blog. https://github.blog/2013-07-02-release-your-software/

Open Source Initiative. (2007, March 22). The Open Source Definition. https://opensource.org/osd/

Open Source Initiative. (2018, October). History of the OSI. https://opensource.org/history/

Open Source Initiative. (2023, February 16). About -. https://opensource.org/about/

Rashid, M., Clarke, P., & O'Connor, R. J. (2019). A systematic examination of knowledge loss in open source software projects. International Journal of Information Management, 46, 104–123. https://doi.org/10.1016/j.ijinfomgt.2018.11.015

Raymond, E. (1999). The cathedral and the bazaar. Knowledge, Technology &Amp; Policy, 12(3), 23–49. https://doi.org/10.1007/s12130-999-1026-0

Resource limitations - GitHub Docs. (n.d.). GitHub Docs. https://docs.github.com/en/graphql/overview/resource-limitations

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), 13–19. https://doi.org/10.22215/timreview/645

Setia, P., Rajagopalan, B., Sambamurthy, V., & Calantone, R. J. (2012). How Peripheral Developers Contribute to Open-Source Software Development. Information Systems Research, 23(1), 144–163. https://doi.org/10.1287/isre.1100.0311

Sharma, P. N., Hulland, J., & Daniel, S. L. (2012). Examining Turnover in Open Source Software Projects Using Logistic Hierarchical Linear Modeling Approach. In IFIP advances in information and communication technology (pp. 331–337). Springer Science+Business Media. https://doi.org/10.1007/978-3-642-33442-9_30

Still, B. (2008). An Open Source Primer. Global Information Technologies, 37. https://doi.org/10.4018/978-1-59904-939-7.ch004

Trending repositories on GitHub today. (n.d.). GitHub. Retrieved April 7, 2023, from https://github.com/trending

Wei, K., Crowston, K., & Eseryel, U. Y. (2021). Participation in community-based free/libre open source software development tasks: the impact of task characteristics. Internet Research, 31(4), 1177–1202. https://doi.org/10.1108/intr-03-2020-0112

Wei, K., Crowston, K., Eseryel, U. Y., & Heckman, R. (2017). Roles and politeness behavior in community-based free/libre open source software development. Information & Management, 54(5), 573–582. https://doi.org/10.1016/j.im.2016.11.006

Xia, Y. (2020). Correlation and association analyses in microbiome study integrating multiomics in health and disease. In Progress in Molecular Biology and Translational Science (pp. 309–491). Academic Press. https://doi.org/10.1016/bs.pmbts.2020.04.003

8 Appendices

A – Year-on-year heatmaps based on the first year project type

Projects that start as:

Low sustainability, high innovation

Sust. / Innov. | Year	1	2	3	4	5	6
Low - High	81	24%	28%	25%	30%	20%
High - High		49%	39%	41%	35%	60%
Low - Low		20%	26%	20%	22%	13%
High - Low		7%	7%	14%	13%	7%

High sustainability, high innovation

Sust. / Innov. | Year	1	2	3	4	5	6
Low - High		15%	19%	17%	11%	13%
High - High	243	58%	48%	53%	47%	45%
Low - Low		9%	15%	14%	19%	25%
High - Low		18%	18%	16%	23%	17%

Low sustainability, low innovation

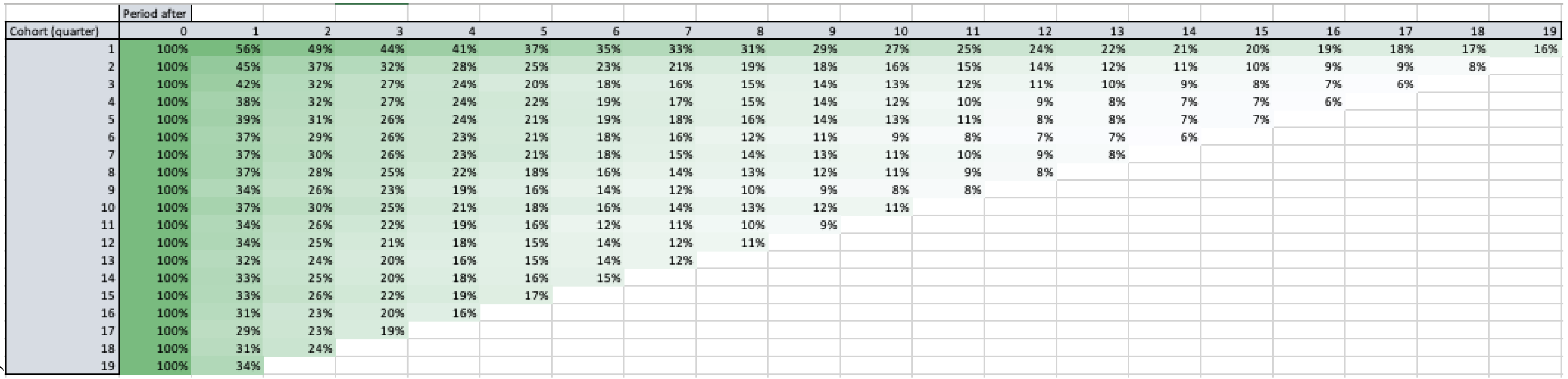
Sust. / Innov. | Year	1	2	3	4	5	6
Low - High		6%	7%	7%	9%	13%
High - High		6%	18%	12%	22%	19%
Low - Low	181	42%	38%	39%	34%	35%
High - Low		45%	37%	42%	35%	32%

High sustainability, low innovation

Sust. / Innov. | Year	1	2	3	4	5	6
Low - High		5%	7%	9%	9%	16%
High - High		14%	12%	17%	18%	15%
Low - Low		37%	39%	37%	34%	37%
High - Low	290	44%	42%	38%	38%	32%

B – Average retention over time, for Microsoft and Google combined

(the values that go over the selected period of 5 years (20 periods) have been deleted to ensure integrity of the results)



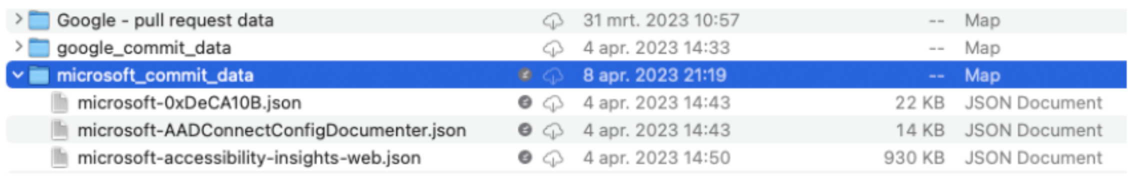
	Period after																			
Cohort (quarter)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	100%	56%	49%	44%	41%	37%	35%	33%	31%	29%	27%	25%	24%	22%	21%	20%	19%	18%	17%	16%
2	100%	45%	37%	32%	28%	25%	23%	21%	19%	18%	16%	15%	14%	12%	11%	10%	9%	9%	8%	
3	100%	42%	32%	27%	24%	20%	18%	16%	15%	14%	13%	12%	11%	10%	9%	8%	7%	6%		
4	100%	38%	32%	27%	24%	22%	19%	17%	15%	14%	12%	10%	9%	8%	7%	7%	6%			
5	100%	39%	31%	26%	24%	21%	19%	18%	16%	14%	13%	11%	8%	8%	7%	7%				
6	100%	37%	29%	26%	23%	21%	18%	16%	12%	11%	9%	8%	7%	7%	6%					
7	100%	37%	30%	26%	23%	21%	18%	15%	14%	13%	11%	10%	9%	8%						
8	100%	37%	28%	25%	22%	18%	16%	14%	13%	12%	11%	9%	8%							
9	100%	34%	26%	23%	19%	16%	14%	12%	10%	9%	8%	8%								
10	100%	37%	30%	25%	21%	18%	16%	14%	13%	12%	11%									
11	100%	34%	26%	22%	19%	16%	12%	11%	10%	9%										
12	100%	34%	25%	21%	18%	15%	14%	12%	11%											
13	100%	32%	24%	20%	16%	15%	14%	12%												
14	100%	33%	25%	20%	18%	16%	15%													
15	100%	33%	26%	22%	19%	17%														
16	100%	31%	23%	20%	16%															
17	100%	29%	23%	19%																
18	100%	31%	24%																	
19	100%	34%																		

C – Data management plan

This data management plan outlines the lifecycle of the data that will be gathered from GitHub's GraphQL API for the purpose of this research. The plan covers how the data will be collected, stored, organized, and analyzed, as well as how it will be shared and preserved after the research.

The data will be collected from GitHub's GraphQL API using appropriate queries. The data will be collected using a personal access token, which grants free access to all endpoints, with a limit of up to 5.000 requests (points) per day. This is done using a script written in Python, that periodically retrieves the commit data from a single project, with 5 seconds delay (sleep) between requests as to not overload the server. After all commits from a project have been collected, they are together exported as a JSON file with the name of the project for further analysis. Then, the script moves on to the next project to repeat the process. The same process will run for all commits and pull requests of a project.

The JSON files are stored in a folder named after the project's respective owner (e.g., Google, Microsoft, or Apache), and type of data (commits or pull requests):



Google - pull request data	31 mrt. 2023 10:57	--	Map
google_commit_data	4 apr. 2023 14:33	--	Map
microsoft_commit_data	8 apr. 2023 21:19	--	Map
microsoft-0xDeCA10B.json	4 apr. 2023 14:43	22 KB	JSON Document
microsoft-AADConnectConfigDocumenter.json	4 apr. 2023 14:43	14 KB	JSON Document
microsoft-accessibility-insights-web.json	4 apr. 2023 14:50	930 KB	JSON Document

This is stored on a personal OneDrive, which has been backed up to both a local machine and an external SSD. That way, the integrity and availability of the data are ensured. Since all collected data is publicly available, and only aggregates will be used (e.g., total committers instead of naming individuals), there is no need to anonymize the data to ensure its confidentiality.

The collected data will be shared with all other parties related to this thesis. That is, it will be shared with Tim van der Heijden (fellow thesis-writer working on a similar, qualitative research), Poonacha Medappa (thesis coordinator and researcher), and Arash Saghafi (co-researcher with Medappa).

After the research is complete, the data will be preserved for future use. The data will again be stored in a secure and backed-up location to ensure its long term availability.