

A theory of FLOSS projects and Open Source business models dynamics

Abstract

In this article, we propose a theory that explains how Free/Libre Open Software (FLOSS) projects work and how companies rely on these FLOSS projects to develop their commercial offers, what we refer to as their "open-source" business model(s). This article builds on and refines the studies of the FLOSS organization by connecting two interrelated aspects: 1) how this organization evolves over time, in order to 2) better understand the value that users create and capture at each moment of a FLOSS project, with a particular focus on open-source companies, which are specific users who do business based on the software created by the FLOSS project. We describe these models and show that the open-source business models of companies are based on contributing to FLOSS projects in order to be able to provide "3A" services (assurance, adaptation, and assistance or support for use) that are complementary to the access to the software. Providing these services requires participation in the FLOSS project, which provides the project with the resources to operate.

This work can help the software engineering community by showing how FLOSS evaluation tools can be improved by taking into account the maturity of the solution, the strategic need of the target user, and the complementary open-source offers that exist.

Keywords: FLOSS, Open-source, Business models, Value network, Software-flow, Strategy

1. Introduction

There is an extensive research on the functioning of Free/Libre Open Software (FLOSS) projects and on how companies rely on them to develop their commercial offerings (see Shahrivar et al., 2018; Liao et al., 2019; Trzeciak

et al., 2022; Li et al., 2024, for recent reviews). These works describe a complex situation where some projects coordinate numerous actors without a clear hierarchy, an organization that has been called a "bazaar", to take the famous work by Raymond (1999), or "FOSS" projects by Fitzgerald (2006) while others seem to be controlled by a single company, such as MySQL, for example, others by a non-profit organization (e.g. a foundation like Linux by the Linux Foundation).

FLOSS projects have first been described as actors (individuals or organizations) working together to develop and maintain a solution they need and use (von Hippel and von Krogh, 2003). In such a project, these "users" who will contribute to the production effort are a priori unknown, and contributing is not a prerequisite for accessing the software¹. The licenses proposed in these types of projects allow for the release of the source code and its free use². The term "free" is used in the sense of freedom of use, but in most cases also implies free of charge.

But FLOSS projects have also served users beyond their contributors, including commercial activities, since as early as the 1990s (Red Hat was founded in 1993 for instance). These offers have found a clientele among organizations³. Today, a FLOSS project can be used by different companies with different business models. Faria and Belfo (2019) detail the model of three Portuguese companies proposing commercial offers based on the same open Enterprise Resource Planning (ERP) Odoo going from selling development times to the billing of the installation and maintenance of complete solutions (in Portuguese).

These open-source commercial offers based on FLOSS solutions, or of FLOSS projects managed by commercial entities, seem to have outnumbered the early collective FLOSS projects, if not made them disappear. This phenomenon

¹We use the vague term "user" to refer to an economic agent that uses a solution, without distinguishing whether that agent is an individual or an organization. When necessary, we will specify which type of user we are referring to.

²It is beyond the scope of this article to discuss in depth the IP aspect of software and of open-source. A good introduction can be found in Thiruthy (2017).

³See, for instance Markets and Markets' 2021 report for details of open-source companies' clients. See the recent special issue of the Journal of Systems and Software for a presentation of the challenges for companies to adopt FLOSS (Stamelos et al., 2020) in terms of reusability but also security, but also the study of Chesbrough (2023) for the Linux Foundation on the massive adoption of FLOSS products by Fortune 500 companies.

have been described as "open-source 2.0" by Fitzgerald (2006). Many studies have described the diversity of the open-source commercial offers, from West (2003); Bonaccorsi and Rossi (2003); Dahlander and Wallin (2006) to Duparc et al. (2022). However, as detailed in the next section, most have focused on a single type of open-source business, namely that in which a company develops and controls a FLOSS project. Yet most of the open-source business models seem to be based on consulting on or facility management of software produced by a FLOSS project ⁴. It also remains difficult to understand how these different business models articulate with FLOSS projects and contribute (or not) to their support (Trzeciak et al., 2022), even if some case studies are quite enlightening on that aspect. For instance Zhang et al. (2021) proposed an in-depth analysis of corporate participation in the Opentack Cloud FLOSS project, which involves hundreds of companies, beside independent developers and non profits. They showed that companies participate in the project in different parts and with different intensity, depending on their use of the software and especially on their offer: "Companies that make substantial and extensive contributions tend to be the providers of a full cloud solution. Usage-oriented companies tend to contribute to a large scope of projects and have a preference for deployment tools. Companies that select specific projects to contribute to either are driven by their particular business or are infrastructure vendors who develop infrastructure that supports the development of OpenStack" (p. 2252). But their results need to be generalized. This article aims to do so by proposing a theory of how the FLOSS projects and the open-source company develop and interact. As exemplified by this last sentence, it will do so by distinguish between the software development project (which will be called "FLOSS" business model) and the use of its production in commercial offers (which will be called "Open-source" business models).

Our analytic framework draws on the "business model" framework (Massa et al., 2017; Pisano and Teece, 2007; Teece, 2018) that has already been used by many studies on the open-source business models (Lin and Maruping, 2022; Trzeciak et al., 2022), even if most of them address only some elements of

⁴In addition to the already cited Market and Market's article, see Kang and Son (2016) in the field of (open) Geographic information systems where they claim that "the service and support business model takes the highest percentage in the business of open source GIS software, and consulting does the second highest".

the business model (see Section 2). This framework emphasizes the fact that an organization (here the FLOSS project or the open-source company) can be studied in terms of its capacity 1) to create value (for its users, here the outputs of the FLOSS project and the commercial proposal of the open-source company) and 2) to capture a part of this value to cover the costs of 3) the resources it has to develop and the activities it has to carry out to create said value.

This definition of a business model helps to highlight several points regarding the functioning of FLOSS projects, but also of the open-source companies that are at the focus of this article. First, although value capture is often seen as a mere financial flow⁵, it must not be reduced to this, as illustrated by open innovation practices, through which some actors can benefit from the intellectual production of others (Chesbrough et al., 2018). FLOSS projects capture value in the form of human time that the various participants devote to the development of a shared code. On the other hand, despite the availability of the code, not all users seem able to benefit from the software solution as they pay open-source companies to "use" it. Second, a business model goes beyond a billing model, or a value creation/value capture model: it is about articulating an offer (what value is created for the "user", sometimes the "client") with how the elements needed in order to propose this offer are developed (value creation)⁶. These elements include the software code, as well as the other elements needed to use it (such as user guides), but also the capacity to match needs with software, and to adapt each one to the other, what (Ethiraj et al., 2005) called the "user capabilities".

This means that the participants in a FLOSS project create and capture value in many forms, depending on the structure of the project, but also on the strategies of these participants: "beyond just saving time and money, non-financial gains such as the knowledge and ideas acquired externally

⁵An example of such a narrow understanding of what a business model is can be found in the analysis of revenue capture in the field of open-source Geographical information system field by Kang and Son (2016).

⁶While Raymond (1999) has been seminal in explaining the benefits for a company to develop an open-source billing model, this work is also an example of the lack of articulation between value creation/capture and the resources. It does not explain why and how much open-source companies must invest in the FLOSS project either, something that research still has to explore today, according to Trzeciak et al. (2022).

proved extremely beneficial in terms of value capture for the companies in this study" (Morgan and Finnegan, 2014, p. 235). FLOSS projects are "value networks", i.e. an "organization of the cooperation of different actors in the production and capture of value that they use to satisfy their needs" (Morgan et al., 2013).

This article proposes to refine the study of the FLOSS organization as a value network by exploring two interconnected aspects:

- **RQ1: how does the FLOSS organization evolve over time?** We will show that software has to be initiated, but also developed, expanded and maintained over time; this will be done by analyzing what users capture when they adopt a FLOSS project's software, what investments are made in the FLOSS project, and how the project organizes these captures and investments.
- **RQ2: among FLOSS users, what value is captured by open-source companies, which do business based on the software created by the FLOSS project, and thanks to what investments made in the project?**

In doing so, this article aims to develop the initial findings of Lin and Maruping (2022), who emphasized "the importance of temporal dynamics and processes in understanding how new ventures organize for open innovation in emerging large-scale collaboration environments" (p. 225). It aims to provide a theory of the economics of FLOSS projects, by developing the "preliminary propositions" of Jullien et al. (2019). Specifically, this article provides a taxonomy of FLOSS and open-source business models, and explains at what point in the life of the FLOSS project each open-source model is possible.

The article is structured as follows. Section 2 draws on the literature to provide the necessary background on the software industry so as to understand how the shortcomings of classic, private software production have fostered the emergence of FLOSS and open-source models. It also points to the limited understanding of their articulation in the literature. Section 3 aims at filling this gap. It examines the evolution of FLOSS projects' business models in terms of organization and benefits to users, and identifies how each stage of the FLOSS business model allows for the construction of different open-source business models by companies. The final section discusses the different FLOSS and open-source business models, which differ in the way in which they

articulate value creation, value capture and the capacities to do so (Baden-Fuller and Morgan, 2010), as well as addressing their articulation, before concluding.

This work theorizes that the main value creation of FLOSS projects involves managing the software "flow" – the evolution of software versions, to develop its functionalities, for security reasons (bug fixes), or for technological evolution – beyond the "stock", i.e. accessing a software product "as is". In order to benefit from this evolution, users need to participate in the production, to see their needs taken into account, but also to develop the skills to be able to contribute or simply to understand how the solution works and can be adapted to their needs, via a mechanism close to the one described by Cohen and Levinthal (1989) for innovation. Because certain users do not have the necessary skills or find it cheaper to outsource this effort, they may pay open-source companies to "participate" for them. Depending on their position in the FLOSS project and the maturity of the FLOSS project, these open-source companies can develop different business models, which are detailed below, but are all based on the offering of "3A" services that complement access to the software (assurance, adaptation, and assistance or support for use). FLOSS benefits from these contributions. Offering these services requires investment from open-source companies, i.e. participation in the FLOSS project, which, in turn, provides the project with the resources to operate.

2. Software Industry, FLOSS Advantages and Open-source Business Models: State of the Art

We first explain how the basic condition of software production has contributed to structuring the industry, but has also created dissatisfaction among certain users, explaining the emergence of the FLOSS movement. By better defining the characteristics of these specific users and of other software users, we explain why there is a market for open-source companies and summarize what is known and not known about them.

2.1. The software industry

2.1.1. Software solutions

Modern software solutions are shared by multiple users for obvious cost reasons. However, because of the variety of the demands, a software solution, and its producer, have to deal with somewhat contradictory goals (Cusumano,

2004; Horn, 2004): to be as adaptable and “customizable” as possible, but also as stable as possible, to be resilient in the long term. Producing software is not like producing a physical product, it is something that has to be organized and sustained over years to deal with the evolution of needs, but also of the technologies with which a software product interacts (other software, hardware, etc.)

Most of the time, the number of competing solutions is reduced, because of the strong increasing returns to adoption effects (in the sense given by Arthur, 1994): economies of scale because of the zero marginal cost of reproducing it, technological interrelations (between different software, such an operating system and a business software solution) as well as, since the Internet, network externalities (software need to exchange compatible data).

To articulate the few solutions with the various needs, most of the software solutions is a combination of a more or less central "software base" that allows economies of scale on the standard part of the needs, and services consisting in requirement specification, solution selection, adaptation, and maintenance in operational condition (Cusumano, 2008).

The possible arrangements to produce such a solution vary, depending on who allocates and pays for the human and technical resources needed to develop two main capacities (Ethiraj et al., 2005): the capacity to translate user needs into technological requirements, and the mastery of the development process (developing, but also prioritizing and integrating the code into a coherent solution). It can be produced through supply pooling (a publisher creates a solution and sells it to "customers") or demand pooling (a group of actors co-develop a solution that meets common or complementary needs), with various intermediate arrangements (actors commission a vendor to develop a solution for them).

In the dominant model that emerged in the 1980s, and which we will call the "classic" software business model, software is developed by a private actor. This does not goes without limitations.

2.1.2. The "classic" software business model

In this classic business model an actor, that we will call a software "publisher", invests in the development of a software, by identifying the needs and coding the corresponding solution. The ownership of the code, granted by the copyright system, allow the owner to select those who are authorized to use the software. The authorization is called a "license" to use "copyrighted"

software code, and is usually granted in exchange for payment. This actor also has to manage the software's evolution over several years, to meet evolving needs and technologies, over and above day-to-day maintenance (e.g., bug fixing) (Cusumano, 2004). The cost of this evolution management is financed by selling access rights to new versions of the same solution or, more recently, by granting renewable access for a limited period of time of the online application, in exchange for more regular updates of the solution⁷.

This structure has its disadvantages. The software evolution remains entirely dependent on its publisher. Any intervention on the software is impossible, even for correcting errors or enriching functionalities. And it is important to remember that while the advent of software packages in the 1980s made the software base part more visible, the service part of selecting, but also customizing/ integrating a solution into an internal infrastructure remains the largest source of spend, especially for business customers (Campbell-Kelly and Garcia-Swartz, 2015; Yost, 2017). It is also easier for this publisher to propose extensions or complementary software, and to prevent competitors from entering or remaining on the market by changing the interface, impeding the dynamics of the solution and its ecosystem –an illustration is the competition between Microsoft and Netscape in web browsers (Gilbert and Katz, 2001) and the lawsuits that nearly lead to the split of Microsoft in two companies (operating system and browser) (Economides, 2001). This explains, with the diffusion of the Internet, the Free Software Movement (Tuomi, 2001), as a specific type of demand pooling organization.

2.1.3. The appeal of FLOSS

Demand pooling, i.e. users dedicating resources (people) to a software project is indeed how the first programs were developed, and a strong culture of cooperative development has long existed in the research community and between public research centers and industry (Rosenberg, 1982). In the mid-1980s the Free Software Movement, initiated by Richard Stallman, a researcher at MIT, relaunched this practice, providing the industry with new standards, such as Linux (operating system) or Apache (web server)⁸.

⁷For instance, Microsoft proposes or the purchase of a perpetual license of its products, for one computer and without update, or access to its cloud offer, on an annual basis. Today, Adobe propose only online access to its solutions, on a yearly or monthly basis.

⁸The original manifesto for the development a completely free operating system, the GNU manifesto, was published in 1985, following an original announcement in 1983. Among

For users capable of coding their needs, the availability of the source code opens up the possibility of making improvements by themselves, something often less costly than waiting for the publisher to make appropriate upgrades (Franke and von Hippel, 2003). Moreover, they have an incentive to share these improvements. They can expect feedback on them, and complementary and cumulative innovations. Their modifications, which reflect their specific needs, can be integrated and maintained in future versions. They also benefit from individual learning effects and reinforce their competence (von Hippel and von Krogh, 2003; Lakhani and von Hippel, 2003). These users have been named “innovative users” by von Hippel who highlights their importance in the initiation of most FLOSS projects (ibid).

These users can be organizations, and West and Gallagher (2006) identified four main strategic reasons for them to do so. (More recent studies, synthesizing the reasons why corporate engage in FLOSS, such as Germonprez et al., 2017; Kendall et al., 2020, detailed these reasons without challenging the framework.) Organizations may seek to create a kind of pooled R&D project (e.g., Kubernetes, a solution “for automating deployment, scaling, and management of containerized applications”, i.e. a tool for cloud deployment, “builds upon 15 years of experience of running production workloads at Google, combined with best-of-breed ideas and practices from the community”). They may spin out an existing software solution that becomes too costly to maintain with regard to its strategic advantage, such as Netscape’s web browser, Mozilla (Ågerfalk and Fitzgerald, 2007) that became Firefox in 2004⁹. More recently, planned FLOSS projects have emerged, usually led by a consortium of companies in order to share development costs around a new industry standard (According to Teixeira et al. (2016), OpenStack can be seen as such an example, even though it was initiated by two actors, Rackspace Hosting, an American cloud computing company, and NASA, but quickly opened up to other participants).

For non-innovative users, too, FLOSS solutions may be cheaper, given the ab-

other things, and as the first FLOSS solutions, Stallman created Emacs (1984), which is still used and among the best program design tools today, and the GCC (1985) compiler for C and C++ (among other languages), which are still widely used. On the subject of the rise of Linux as an alternative standard to proprietary operating systems see also West and Dedrick (2001).

⁹The Mozilla foundation has managed this project since 2003.

sence of fees, but also due to better compliance with standards/norms that facilitates interoperability, thus integration in the information system (Almeida et al., 2011; Sánchez et al., 2020). It is never a zero cost solution, because of the cost of this integration into the information system, something not always anticipated by these FLOSS solution adopters (Morgan and Finnegan, 2014). Olson et al. (2018) detailed the different challenges and costs of such an integration in the case of Enterprise Resource Planning (ERP), and Butler et al. (2022) addressed the general case of choosing a FLOSS component. Non-innovative users do not have the capacity to develop their needs into the FLOSS project. Must they find an actor to do it for them to be able to use the FLOSS solution, and is that the core business of open-source companies? Can this solution be used without contributing and for what needs (after all, not all innovative users contribute to the FLOSS solutions they use)? To answer these questions, we need a better definition of what is behind the choice of a software solution, in terms of users' needs and skills.

2.2. Choosing a software solution

A software solution is a resource whose choice depends on (Shaikh and Cornford, 2011): its level of strategic importance as a resource (in the sense given by Grant, 1991; Teece, 2010), but also on the capacity to evaluate and appropriate the solution (Ethiraj et al., 2005).

2.2.1. Characterize the users' requirements for a software solution

A software solution's strategic importance is evaluated in terms of "software-stock" (is the resource more or less specific at the time of acquisition?), but also in terms of "software-flow" (should it evolve over time?) (Shaikh and Cornford, 2011). The more strategic and durable the resource for the company, the more important its "upgradability" is, i.e. the fact that it will be maintained (but fixing) but also improved to adapt to new technologies or needs (ibid). The more the software needs to be adapted to the needs (in terms of functionality or integration with the infrastructure), the more important its adaptability is (Bennett et al., 2000; Li et al., 2009).

"Upgradability" and "adaptability" lead to four models of needs (Figure 1). If the resource is perceived as being unlikely to change, it is what we will call a "commodity", as defined by Van der Linden et al. (2009), and can be served by what has been called "Off-the-self" solutions (Li et al., 2009; Van der Linden et al., 2009), from commercial or FLOSS origins. If little customization is

expected, one will try to acquire a standard solution at the best cost. We will call this "Low-cost" needs. Browsers, whether FLOSS (Firefox) or proprietary (Edge, Opera), or office suites (Libre Office, MS Office) are such examples of commodities that serve low-cost needs. If a lot of customization is needed, one will look for an adaptable solution at the best cost. We will call this "Adaptable" needs. An example of such needs and solutions is the Enterprise Resource Planning offers (SAP ERP, Oracle ERP, Microsoft's Dynamics 365 are proprietary examples, Odoo, ERP5, or Dolibarr are FLOSS examples). The resource may have a significant upgradable dimension, due to expected technological changes or expected evolution of internal needs. If the need is not too specific, users will look for a solution that guarantees technological compatibility over time, and avoids lock-in as much as possible (what we call "Lock-out"). For example, a network infrastructure (hardware plus software) must comply with standards such as IPV6 to allow interconnection with the outside world. This can be provided by private companies, such as Cisco, or by FLOSS solutions such as those presented by the TekLager company, which implements FLOSS routing operating systems such as OPNSense, pfSense, or OpenWRT. If the need is seen as both specific and evolving over time, users will consider whether they can "control" the adaptability of the solution over time. The type of solutions involved is highly dependent on the companies, as emphasized by (Van der Linden et al., 2009), infrastructure systems are an example, such as systems for automating the deployment, scaling, and management of containerized applications¹⁰.

However, this choice also depends on the users' capacity to make it. Software is a highly technical industry, in which the users' capacity to envision the technical needs and master the translation of business (functional) needs into (software) specifications is at least as important as the capabilities of solution providers to offer an effective solution (Ross et al., 1996).

2.2.2. Characterize the user's ability to choose a software solution

The two capacities defined before with Ethiraj et al. (2005) (to translate needs into technological requirements, and to develop –code – them) allow to split software users into three groups (Jullien and Zimmermann, 2011; Visieur and

¹⁰most solutions are based on the FLOSS Kubernetes, but there are company offerings that complete them with private add-ons, as explained by TechTarget.com in the article "Compare 10 leading container management systems", published online September 30, 2019.

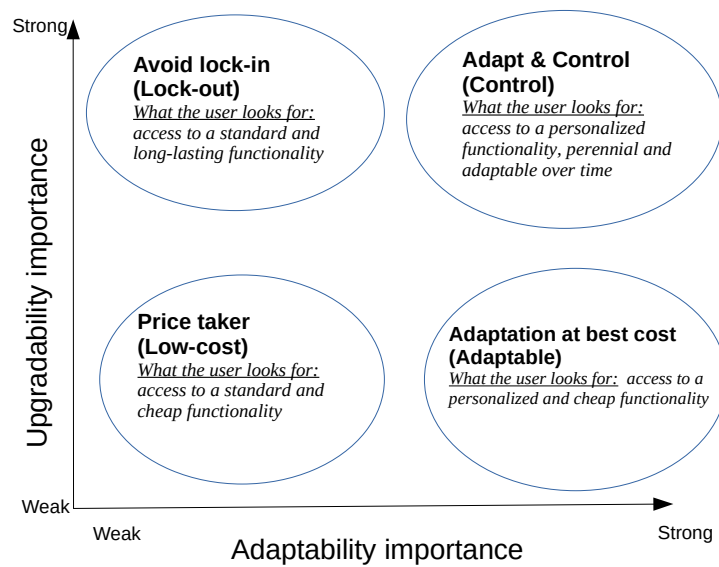


Figure 1: Segmentation of users' needs according to the characteristics of their requirements for a software solution

Jullien, 2023) that are summarized in Table 1. The first group masters both skills and is thus able to translate program requirements into code. These are the above-mentioned "Innovative-users", as they are referred to by von Hippel (2001). The second group is capable of translating needs into functional requirements or finding bugs in programs, but not participating in their development. Kogut and Metiu (2001) stressed their importance for software production because they help define the needs and verify the quality of the solution, but remain at the border of the development project. In accordance with these authors, we will call them "Frontier-users". The third group, characterized by the absence of both capacities, are called "No-skill-users".

Table 1: Type of software solution users according to their capacities. Based on Ethiraj et al. (2005); Jullien and Zimmermann (2011); Viseur and Jullien (2023)

	Capacity to translate needs into technological requirements	Capacity to develop – code – the needs
Innovative-user	Yes	Yes
Frontier-user	Yes	No
No-skill-user	No	No

The most competent users do not necessarily outsource the least; they are more able to define their needs, and to assess the quality of the provider (Tadelis, 2007; Weeks and Feeny, 2008; Gambal et al., 2022; Simões and Santos, 2024). Innovative-users have access to a larger strategic portfolio of solutions, between those they may implement themselves and those they may outsource, than Frontier-users, who have to outsource the development of their requirements. However, Frontier-users are able to define, specify their needs and evaluate solutions accordingly, while No-skill users have limited exploration capabilities of the various types of offers available. Open-source companies can provide services of specifying the needs and evaluating the (FLOSS) solutions according to these needs (No-skill users), but also by developing the needs and getting them accepted by the project (for all the users). However, we will now show that it is not what has been the most studied in the literature on open-source business models.

2.3. Open-source business models: what is known

A search in Scopus and the Web of Science using the keywords "open-source" and "business model" yielded 173 articles, starting with Hecker (1999). Some of them are loosely connected to the “business model” aspect but address other interesting aspects, such as security advantages, or implementations. We completed this search with querying “open-source business model" on SciSpace, without any further additions. We completed this list with readings about FLOSS project organization, which, surprisingly gave older literature (Google scholar research, keywords “open-source communities”), and with a search of the newest articles those found through the first two searches. After reading all the abstracts, we based our analysis on the most recent articles (published in 2018 or later) in addition to the references of the field (evaluated by our knowledge of the field and the number of citations). We also completed the description of our business models by identifying references to the selected articles, via a Google Scholar search for papers citing the articles identified in the literature review. We ended with 147 articles cited here. We summarize below what these articles have explained and why we see a need to complement their findings. Several articles proposed a literature review in this field (Shahrivar et al., 2018; Li et al., 2024, are the most recent).

The recent literature on FLOSS projects focuses on project organization and governance and the relationship between open-source companies and projects, trying to better understand the value of these companies’ contributions to the

FLOSS project, but also the risks of these contributions in terms of project stability (Ferraz and Santos, 2021; Viseur and Charleux, 2021; Schaarschmidt, 2023), as well as the link between open-source companies' participation in FLOSS and their commercial offer (Mouakhar and Benkeltoum, 2020).

Regarding the open-source aspect, some articles proposed either a thorough analysis of up to 120 FLOSS projects/open-source companies (Duparc et al., 2022), or a smaller number but trying to focus on one type of commercial offer (Spijkerman and Jansen, 2018, ten cases of B2B open-source companies studied). Other articles focused on a detailed analysis of a specific ecosystem (Faria and Belfo, 2019, on the business models of open-source companies implementing Odoo), a specific market (Kang and Son, 2016, on the geographic information systems field) or a specific open-source business model (Riehle, 2021, on the "distributor" business model, what we will refer to as "publisher" open-source business model).

In most of these studies, the FLOSS project is seen as a resource with which some of the software development can be outsourced, and which is then used to build a classic offering (West, 2003; Bonaccorsi and Rossi, 2003; Dahlander and Wallin, 2006; Chesbrough and Appleyard, 2007; Kang and Son, 2016): selling licenses to use complementary software products or hardware, or selling deployment and maintenance services¹¹.

The open-source business model most studied, and recently reviewed by Shahrivar et al. (2018), is the "open-source editor", when an actor manages a FLOSS project for which it does most of the development. We assume that this is due to the fact that it is the most antagonistic to the classic model of the private software editor that sells access to its intellectual property. Several business models have been identified, which differ in terms of what is charged to customers and the degree of involvement in the FLOSS project.

These studies helped us identify different strategies to propose value and capture it in the context of an open-source business model, or the participation of open-source companies in a FLOSS project, and, as previously mentioned, we will refer to them as illustrations of the different business models we present below. However, they do not provide a complete picture of the business model

¹¹VMWare's Tanzy Platform offering in the field of container orchestration is an example of such a proposition: "Tanzu Labs provides services for every step of your platform journey, from configuring your platform and migrating valuable applications to helping you implement best practices for managing a developer-centric platform."

of the FLOSS project and of these open-source companies, as defined in the introduction: the organization's capacity to 1) create value (for its users, here the outputs of the FLOSS project and the commercial proposal of the open-source company) and to 2) capture part of this value to cover the costs of 3) the resources it has to develop and the activities it has to carry out to create said value. Most of the time, point 3 is missing in the business model articles, while Demil and Lecocq (2010) stressed that value capture is underpinned by a dynamic articulation between value propositions and resources, and, as already said, Zhang et al. (2021) exemplified this in the case of OpenStack for FLOSS.

To get a more complete picture, it is necessary to start from the FLOSS project, and more specifically, to understand why some software users have chosen to collaborate with others in an open way to produce software (Trzeciak et al., 2022). As explained by Morgan et al. (2013), the core of a FLOSS project is the building of a "value network", which needs to be studied, i.e. why potential complementors are crucial to the participants for their value creation/capture, but also their level of commitment to the project: the effort they put into it, how they collaborate to align their goals with those of other partners in this project.

We will show that each phase of the FLOSS project corresponds to a specific organization, a specific FLOSS business model, but also specific companies' open-source business models. We will present these phases and their business model, and the elements of their evolution, using the business model canvas (Osterwalder and Pigneur, 2010) that allows for a synthetic description of the different elements of a business model. Presented as a table of nine cells, it details the different elements of value creation, value capture, and resources (see Table 2). The proposition of value is detailed in *Value proposition*, but also to whom (*User segmentation*), how it is delivered (what is exchanged with the users, or *user relationships*) and through which *Channels* (platform, physical platform...) The resources to do this are detailed in *Key resources* (what the organization owns that allows it to deliver the value), but also *Key partners* (with whom they build the resources and the proposition of value. For open-source companies we will discuss if/how the FLOSS project is a key partner) and doing what action (or *Key activities*). Finally, the value flow is broken down into *Value capture* and *Cost structure*. It is important to understand that value can be monetary, but also in terms of effort (e.g. time spent).

Table 2: Business model canvas, adapted from Osterwalder and Pigneur (2010)

Key partners Key stakeholders for the project/company to build the value proposition. For open-source companies we will discuss if/how the FLOSS project is a key partner	Key activities Activities that are at the heart of building the value proposition	Value proposition What the solution delivers (here a software solution, but sometimes complemented by other software or services)	User Relationships The type(s) of offer (service, product), but also how it is delivered (e.g. the level of personalization).	User Segments Types of users interested in the solution in terms of profile (end user, organization...) but also skills (here: innovative, frontier, no-skill users)
	Key resources The human and technical resources (capital and labor) required to perform the (key) activities and deliver the solution		Channels Channels used to distribute (Online, Stores, On-Premises...)	
Cost structure (time & money) All costs (effort —time—, money...) of the (key) resources and key activities		Value capture (time & money) All flows of value captured by the project or company that "finance" the costs (time or money given by participants, money paid for a product/service sold, etc.)		

We will use this presentation which is the most familiar when presenting a single (FLOSS/open-source) business model. When presenting multiple business models, the nine items are presented in a single column.

3. Understanding FLOSS projects' Business Models to Identify Companies' Open-Source Business Models

We will study how and what value is created by the FLOSS value network, over time, before looking at the open-source models.

3.1. FLOSS projects' global characteristics

As for software projects in general, FLOSS projects are very heterogeneous, in terms of diffusion, the number of people involved, and the quantity of code produced (Krishnamurthy, 2002; Healy and Schussman, 2003; Liao et al., 2019). A minority of these projects aim at providing complex software for complex needs (e.g. operating systems such as Linux) and last more than five years (Ait et al., 2022). The majority address a limited need, manageable by a small team, sometimes reduced to one single developer (Liao et al., 2019). In such cases, however, they cannot be referred to as FLOSS, cooperative projects, and we will see that they lead to limited possibilities of open-source business models in addition to jeopardizing their sustainability¹².

Some constants exist, in terms of project modularity, and specialization of the contributors (Crowston et al., 2008; Medappa and Srivastava, 2019; Palazzi et al., 2019; Joblin et al., 2023): each project is led by a core group of developers (the 'kernel') which develops the main part of the source code, and whose members, or at least some of them, are at the origin of the project (for the majority of the projects, this core team is reduced to one or two developers). A larger group follows the development, sometimes reports bugs, proposes corrections ('patches') or new developments. An even larger group simply uses the program and may sometimes posts some questions on user mailing lists. The bigger a project is, the more complex and strict the mechanisms are to select contributions and questions, so the main developers

¹²The risks, in terms of security (bug correction) and sustainability of such projects maintained by one single person are well explained in Glen S. Small's "open source projects pose significant security risks." TechTarget.com, published online 28 November 2022

are not inundated with peripheral problems. This has been referred to as an “onion model” organization (Figure 2)¹³.

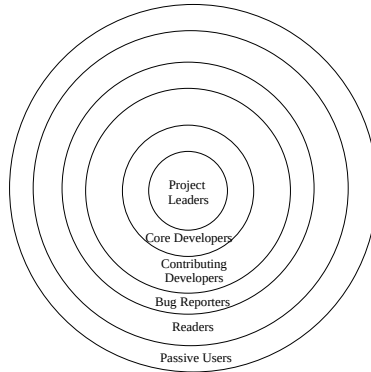


Figure 2: The Onion Model Organization of a FLOSS project, from Amrit and Van Hillegersberg (2010)

Software development management platforms, such as GitHub or GitLab today, allow the core group to accept, or refuse, contributions (for details of such an organization, see Zimmermann, 2020). The “codebase” is designed to comprise (1) a core and (2) a set of modules. As stressed by Moon and Howison (2024), “both academics and practitioners have considered modular (or loosely coupled) software structures to be key to the success of open source software (OSS) projects” (Baldwin and Clark, 2006; Crowston et al., 2008; Lindberg et al., 2016; Medappa and Srivastava, 2019). Modules are “distinct parts of the larger system, which can be designed and implemented independently as long as they obey the design rules” (Baldwin and Clark, 2000). This allows the developers to focus on the parts of the code they are interested in. It reduces the cost of testing new features, too (a change is confined to the affected module and should be tested only there). As explained, contributors have an incentive to make the extra effort to submit their code to the global project, to avoid the risk of certain decisions or a coding evolution making their change incompatible, requiring them to redevelop it. Finally, learning about this organization, being recognized by the core contributors implies, as has been shown for innovation (Dasgupta and David, 1994), having internal resources (developers) to do so (Lin and Maruping,

¹³It is hard to trace the origins of this model. It seems to appear in the analyses of Nakakoji et al. (2002).

2022). In general, it seems that to benefit from the project, companies have to have a "gatekeeper, who navigates code and information flow between his/ her firm and external actors" or the FLOSS project (Nguyen Duc et al., 2017, Subsection 4.1). In concrete terms, it means contributing to the FLOSS project, as this helps to decrease the cost of contributing (by learning how to do so, but also being recognized by the other contributors that makes code inclusion easier (Nguyen Duc et al., 2017, Subsection 4.5)), and for most involved, gives rights to control others' contributions, and thus the project's evolution (Schaarschmidt, 2023; Dahlander and Wallin, 2006; Herstatt and Ehls, 2015, third part). It structures a hierarchy regarding the control of the FLOSS project participants and evolution (Table 3).

Table 3: Roles and power within a FLOSS project. Adapted from Crowston and Howison (2005); Nakakoji et al. (2002)

	Owner	Core contributor	Package manager, main developer	Contributor	User
Access to the software-stock	X	X	X	X	X
Access to the production system (possibility of adding a code that addresses specific/own needs)	X	X	X	X	
Contribution and contributor management (software-flow control)	X	X	X		
Project evolution management (project roadmap control)	X	X			
Alienation management ¹ (brand, code, test bases...) (solution contributors/ users control)	X				

¹ Alienation in the legal sense of the term, i.e. the transfer of intellectual property rights, definitions of the contracts based on the IP ownership (software license, contributor contract, brand use...)

These mechanisms depend on the maturity of the project (Crowston and Scozzi, 2002), and are built in three phases (Koch, 2008; Guimarães et al., 2013; Kuwata and Miura, 2015; Jullien et al., 2019; Liao et al., 2019; Setia et al., 2020): creation, development, and maturity¹⁴. This is not surprising given that these phases are those of software development (Lehman, 1980) and of collective action (Marwell and Oliver, 1993) in its production aspect, and of the diffusion of innovation (Rogers, 2010) for the adoption aspect. We will show that for each phase the FLOSS project proposes and captures different values, but also requires different resources to benefit from them; as a consequence the open-source business models are also phase-dependent.

3.2. Phase 1: Creation. The Innovative-user phase

3.2.1. FLOSS projects: initiating a new development to meet Innovative-users' control-needs

It is difficult for a project to succeed, or even to start, when the initial contributors are not its (first) users (Jullien and Roudaut, 2012). Some Innovative-users have control needs (Figure 1) and perceive the possibility of developing a software solution to meet their needs (thanks to the open contribution mechanism) over time (thanks to the open license mechanism). As explained above, this is understandable when the software solution is a complementary asset (Van der Linden et al., 2009): potentially sharing the same solution with competitors does not threaten their competitive advantage, which relies more on their capabilities to internally tailor and use the solution. The OpenStack project illustrates this: it develops “a set of software components that provide common services for cloud infrastructure”, i.e. the capacity to manage hundreds of computers. It was started by a user, NASA, and is used by many large companies (Adobe, Volkswagen, Walmart, etc.), as well as by some major cloud service providers (Google, OVH-Cloud, etc.)

Unlike an internal project, the FLOSS structure helps share development costs, and multiply the sources of innovation (Lee et al., 2020). However, open to anybody does not mean to everybody. When a project begins, it

¹⁴Kuwata and Miura (2015) speak of five phases. While the correspondence is not perfect, they agree on the idea of phase, and we estimate that the birth and childhood phases correspond to the creation phase of the others, the Adolescence and Adulthood to development, and the end of adulthood and obsolescence to the maturity.

is not very visible (there are millions of projects on these platforms), and, of course, participation requires development skills. All these factors allow the initiators to control the foundations of the project that is developed by a small and very connected group of developers (Joblin et al., 2023).

We see this phase as close to a classic demand pooling consortium, as few actors cooperate to share the cost of developing a dedicated solution. However, there is no need for long negotiations about who does what, who owns what (in terms of intellectual property, for instance), making it easier to integrate new participants. Finally, the fact that a software can be designed incrementally, with a modular architecture that allows each actor to develop its own needs, means that the effort (the investment) can be small and the results can be evaluated almost immediately before deciding to make new efforts. All this is an advantage when starting an innovative (and therefore uncertain) project, because it reduces the risk taken (i.e. the loss of the investment made) (Bessen and Maskin, 2009; Farrell, 1995).

Table 4: Business model for a FLOSS project in phase 1

Key partners Innovative-Users	Key activities Software development	Value proposition Controlled & Personalized software solution	Customer Relationships Customers are producers	Customer Segments Innovative-Users
	Key resources Development platform, Project manager		Channels Development platform	
Cost structure (time & money) Development time (development and project management)		Value capture (time & money) Developers' time		

The project’s business model is summarized in Table 4. Value is created by those who benefit from (and capture) the value created (the software) thanks to the resources they invest in the project (development time). The key capacities are to be able to program the needs, in a collective, open way. This last element is quite specific, and numerous studies have shown that it has to be learned (Stol et al., 2014). This paves the way for a first open-source business model.

3.2.2. Open-source business model: customized software development services

In this phase, the open-source business model is the provision (and billing) of development time, but also of the capacity to understand, and sometimes manage, a FLOSS development (De Laat, 2005; Viseur and Jullien, 2023). Unlike the classic bilateral relationship found in custom development, the small but open group of user-developers must be organized and coordinated without formal hierarchical relationships. Since its purpose is to develop the FLOSS project, the most important resource of the open-source company is the developers it employs. They must be multi-skilled in the sense that they must be able to develop on-demand technical solutions from (Innovative-)customers’ specifications but also to integrate them into a coherent software architecture. This will be referred to as an “*open-source customization*” business model (see Table 5).

Table 5: Open-source customization business model

(Key partner: FLOSS project) Innovative-Users contributing to the FLOSS project	Key activities FLOSS Software development FLOSS project participation	Value proposition Custom development within a FLOSS project FLOSS project management	Customer Relationships Technical, service based	Customer Segments Control (Innovative-Users)
	Key resources FLOSS development experts		Channels One-to-one B2B services via the project	
Cost structure (time & money) Human resource: FLOSS Developers (software development and project management experts)		Value capture (time & money) Billing for development hours		

Based on people's time provision, it involves few economies of scale (even if there are increasing returns on the development skills and on the coordination of different teams/projects)¹⁵.

If the solution matures, new users are attracted¹⁶. Opening up the project beyond the closed club of initiators marks the transition from Phase 1 to Phase 2 (Setia et al., 2020).

3.3. Phase 2: Flourishing development or decline

A project can remain small and be developed and managed by its initiators; as said at the beginning of this section, this is the case with the majority of projects. However, there is a risk that a competing project will take the lead and make it irrelevant until it is abandoned by its promoters-(Ehls, 2017). But growth has its own challenges.

3.3.1. FLOSS projects: structuring open collaboration

The project will grow first with new Innovative-users attracted by its potential, which allows them to exercise some level of control over its future. As the actors diversify, their goals may diverge, too. The difficulty in maintaining technical coherence, as well as the difficulty for newcomers to contribute to the code, may also grow. This is true for any software project, but has specific consequences for FLOSS¹⁷.

As the software industry has gradually evolved from a craft activity dependent solely on the talent of programmers to one of greater industrialization (Duvall et al., 2007), a FLOSS project must be structured (Ferraz and Santos, 2021).

¹⁵It is actually hard to find examples of such activity, as it concerns the beginning of a project, and thus projects that are less visible, and less studied. However Viseur and Jullien (2023) give an example of how the initiators of a project, CommunesPlone, today Smartweb, a Content Management System for municipalities, have been rapidly paid to develop the project for the first users. Another example is the Eclipse Foundation's offer to help actors federate around new FLOSS projects, such as Oniro, an "Operating System Platform for Connecting Devices Big and Small"

¹⁶While making an emerging project visible, attracting new contributors is quite a challenge. The embeddedness of the project's initiators in other projects (Sutanto et al., 2021), and their outreach capabilities in general (Weng and Soh, 2023) seem key here.

¹⁷For example, Daniel et al. (2013), in a study of 357 SourceForge projects, showed that the diversity of contributors' cultural backgrounds has a negative impact on community engagement, whereas diversifying the roles taken in the project, i.e. making the project more structured, has a positive impact on its dynamics.

From a governance perspective, this means finding the right balance between a hierarchical organization, which is a source of productive efficiency, and openness, which facilitates the integration of new members and thus innovation¹⁸. This requires a certain bureaucratization of the organization (O’Mahony and Ferraro, 2007), such as the setting of rules for contributing, and in general a complexification of the governance/coordination framework (Shaikh and Henfridsson, 2017; Palazzi et al., 2019). This structuring of the organization relies on human capacities and involvement (software project developers), and increasingly, as the project grows, technical capacities. In addition to development platforms, other tools are becoming essential such as: bug tracking and bug management systems, non-regression test base¹⁹, feature request management systems, etc. Each tool provides possibilities of managing different levels of contribution rights, according to the centrality of the contributors in the project, their past contributions, and their recognized skills, but also different structures of coordination and governance (Shaikh and Henfridsson, 2017). From a software engineering point of view, it also means working on the architecture of the code, to make it modular²⁰.

We see this as creating a dual movement regarding the possibilities and the incentives to participate. It increases the barrier to contributions relating to the core of the software, because of the cost of learning how it works (for

¹⁸We refer, of course, to the classic dilemma between exploration and exploitation of a technological solution, coined by March (1991). For software, this means a trade-off between increasing the team (and the contributions) and managing the organizational and technical complexity of a bigger project Tilson et al. (2010); Lee et al. (2020); McIntyre et al. (2021). The study of Liu et al. (2021) on software consortium organizations, also strongly echoes our points, in a (slightly) more closed environment.

¹⁹Automating software use scenarios to verify that modifications have not generated bugs.

²⁰The importance of a modular architecture has been stressed by Bessen (2006); Baldwin and Clark (2003). In an early study of the Apache web server project (Mockus et al., 2000), it was hypothesized that open-source projects’ core teams tend to consist of no more than 15 persons, for accessibility and managerial purposes, and this has not been proven wrong since. Well-decomposed OSS architectures are still considered to reconcile the division of labor through self-selection and self-assignment (Palazzi et al., 2019; Shaikh and Vaast, 2023). Lee et al. (2017), in a study of Python, showed that, to increase productivity, most contributors have to individually contribute only to a small number of modules, while a small amount number have to participate in a lot of modules, to facilitate the coordination between sub-modules.

newcomers), but also because of the layers of controls which “protect” against the entry of too many people. This keeps the core team small and makes it possible to continue to manage that core.

As a consequence, the project’s business model does not change much in this phase (see Table 6): users are still contributing to the components they are interested in. However, the contributions, and the skills provided are more heterogeneous, from the core contributors managing the software core to peripheral ones reporting bugs or requesting functionalities (potentially accessible to the Frontier-users), through module developers.

Table 6: Business model for a FLOSS project in phase 2

<p>Key partners</p> <p>Innovative-Users and open-source companies for both the core & the modules (not always the same users)</p>	<p>Key activities</p> <p>Software development project on the core Module development and coordination</p>	<p>Value proposition</p> <p>Modular open software solution (Control/Lock-out) Adaptation to specific needs of a stable solution (Control/Adaptable)</p>	<p>Customer Relationships</p> <p>Customers are producers (directly or through open-source companies) Download of the solution</p>	<p>Customer Segments</p> <p>Core solution: Innovative-Users and Frontier-Users represented by open-source companies</p>
<p>Key resources</p> <p>Development platform, Project manager, Project editor Module developers</p>	<p>Channels</p> <p>Development and distribution platform</p>	<p>Specific needs addressed by specific modules: Innovative-Users and Frontier-Users (represented by open-source companies)</p>		
<p>Cost structure (time & money)</p> <p>Human resources: Developers time (development and project management including module compatibility mgt) Assets: Software infrastructure (online free solution such as GitLab can be used)</p>		<p>Value capture (time & money)</p> <p>Developers' and users' time, including Open-source companies' contributions (paid by users). Some module users contribute to the core to maintain compatibility. Contribution to module development</p>		
<p>In black the business model for the core project In gray the business model for the peripheral module projects</p>				

If the project does meet the challenges of growth, it will have built a specific kind of "architecture of participation" (MacCormack et al., 2006), the “onion” based socio-technical organization we introduced at the beginning of this section (Figure 2). This makes it possible to address "lock-out" requirements on a limited functional basis, because it means that there is a basic solution supported by a working organization.

Later on, and if this modular organization effectively leads to the development of software functionalities, by growing the library of modules, "Adaptable" needs start to be addressable (the core solution with customization or specific peripheral modules). In return, the more contributions there are, the greater the need to verify their coherence, compatibility, and orchestration (See Bogart et al., 2021, for a detailed analysis of the organizational and technical challenges to organize and maintain over time the coherence of project, especially the compatibility between the core and the various modules, referred to in this article as "packages"). This FLOSS project’s new business model opens the door to new open-source business models.

3.3.2. Open-source business models: editor business models and periphery business models.

Different editor business models. The term “open-source editor” introduced earlier emphasizes that, in many projects, one actor coordinates contributions (in terms of feature requests and code), selects them, sometimes develops them, and manages the project roadmap. The larger the project, the more important this role is. There are clear increasing returns that explain why this role is usually played by one actor. First, an editor can amortize a project management platform over several projects: the technical platform, which is less a key resource today with the existence of cloud services such as GitHub or GitLab; the legal apparatus to manage contributions and protect the official distribution of the software, such as the trademark that identifies the project; and the knowledge, embedded in the staff, of managing a FLOSS project. The editor can also build part of the project management system in-house, such as the non-regression test base, a key element to validate the system and ensure it functions correctly for users/clients (for example, Brosgol, 2019, explains how the mastering of a non-regression (private) test base appears to be key to the success of the AdaCore company’s business model). Finally, it can act as the legal entity representing and defending the project’s intellectual property (the project’s code, name or logo) and its

openness²¹.

The open-source editor must finance these editing activities. Several business models can be found, depending on the needs and capacities of the customers addressed (these models are detailed in Table 7 at the end of this paragraph).

The first model, when the editing role is internalized by one Innovative-user, is not really an open-source model: there is no commercial business directly created by the editing activity. This is what Li et al. (2024) described as the "usage-oriented [business] model" and described (p. 17). Ågerfalk and Fitzgerald (2007) explore the concept of "opensourcing", illustrating why a user may wish to pay the costs of managing a FLOSS project: to both benefit from external contributions (new ideas) and keep control of the solution. Two machine learning library FLOSS projects exemplify this: PyTorch, initiated by Meta and Tensorflow, by Google. Both companies needed to develop their machine learning capacity, as complement to their key resource, the data they own. This meant facilitating the transfer of new machine learning tools from the academy and the testing of these tools, which justified their open licensing.

At some point, the user-editor may want to hand over the editing role to another actor, to encourage innovation, or because of the management costs, as did OpenStack initiators in 2012 with the launch of the independent (OpenInfra Foundation)²². Since September 2022, PyTorch is managed by the PyTorch Foundation, a subsidiary of the Linux Foundation, to develop a "vendor neutral" project (Tensorflow is still managed by Google, also a major sponsor of the PyTorch Foundation).

This second model, probably the most original and yet the least studied, with the notable exception of the synthesis proposed by Izquierdo and Cabot (2018),

²¹The importance of a legal entity has been stressed by the creation of the Free Software Foundation and by the study of O'Mahony (2003), for instance. Some projects strive without one. The "PostgreSQL Global Development Group", which is not a legal entity, still manages PostgreSQL, one of the main competitors of Oracle's databases (Oracle and MySQL). However a dedicated association, the PostgreSQL Community Association of Canada, manages its trademark.

²²"OpenStack Launches as Independent Foundation, Begins Work Protecting, Empowering and Promoting OpenStack". BusinessWire. 19 September 2012. Retrieved 10 January 2025.

is the “*foundation*” open-source business model. Many of the best-known FLOSS solutions are managed by such an editor: Linux (Linux Foundation), Apache (Apache Foundation), Eclipse (Eclipse Foundation), OpenStack (OpenInfra Foundation)... These foundations play the role of the neutral actor required to ensure the independence/neutrality of the project, i.e. to manage a consortia of actors who do not want to see the software project fall under the control of a competitor, but need to ensure that it meets their needs ("control" or "lock-out" needs for Innovative-users). They also provide both a technical and also a governance framework that helps the project structure, as shown by Chakraborti et al. (2024) in their study of the Apache Foundation’s project incubator or by the guidelines on starting a project provided by the Linux Foundation. As the OpenInfra Foundation’s membership page shows, foundations charge for this service by proposing different levels of membership, with increasing access to control over the project’s development with the amount paid.

The success of the project also enhances for-profit open-source business opportunities (See Table 7 at the end of the paragraph).

The *open-source customization model* that began in Phase 1 can continue into Phase 2: a core contributor helps some Innovative-users with very complex problems (Riehle, 2012). For instance, the Feexian company employs several Debian operating system core contributors to provide support services to key accounts, in particular deployment and maintenance services for (very) large-scale distributions (facilitated by their knowledge of the package management tools)²³.

With the emergence of more standard needs (Adaptable, Lock-out), come new demands, mainly from Frontier-users, for which the editor has a competitive edge thanks to its technical development infrastructure and its core developers: the provision of additional guarantees with respect to the free offer in terms of solution stability and security. The editor can provide (and sell) two types

²³Another example of such a company is Ideas on Board that advocates its expertise by showcasing its commits to Linux, or Collabora. With it 150 employees, Collabora is "a global consultancy, specializing in delivering the benefits of Open Source software to the commercial world. We enable our clients to develop the best solutions, whether writing a line of code, or shaping a longer-term strategic software development plan. By harnessing the potential of community-driven Open Source projects, and re-using existing components, we help our clients reduce time to market and focus on creating product differentiation".

of certification.

Editors can implement a system to certify the modules (assurance needs), and/or the actors able to develop, implement and maintain them (see the peripheral business models below). The certificates are clear signals of trust for Frontier-users, and open up the market for complementary services. Depending on the project's structure, this "*ecosystem manager*" open-source business model can be performed by a private editor (see the ERP Odoo's certified partners) or by a foundation (e.g. Linux Foundation's Kubernetes certificates), and is today one of the FLOSS Foundations' major activities (Germonprez et al., 2020).

Editors can also provide guaranteed/assured editions of the software solution (e.g. Canonical that offers guaranteed versions of the Linux based operating system Ubuntu). They rely on the economies of scale which arise due to automated integration of a large set of modules that address standard needs. These solutions are sold to clients that require less adaptation (Lock-out), but also to clients less willing to pay, for instance those in need of a stable version of the core software to support peripheral modules that are adapted to their needs (Adaptable, see below). This "*open-source publisher*" business model, described by Riehle (2021), does not benefit from revenues linked to the sale of access licenses in the same way as the classic model, but rather to assurance subscriptions relating to the evolution of the software. The rise in pay-per-use or yearly renewable licenses by companies such as Adobe illustrates the convergence between the two models. Although the high coordination costs cannot be overlooked, open-source publishers benefit from lower development costs than classic publishers, as long as the Innovative-users keep contributing to the development.

Table 7: Open-source Editor business models in Phase 2: value addressed, captured and resources (each business model is presented in on column).

	Customization	Open-source Editor business models		
		Foundation	Ecosystem manager	Publisher
Value proposition	Custom development within a FLOSS project	Participant-Neutral Management of a FLOSS project	Management & certification of FLOSS actors/modules	Provision of a standard maintained FLOSS solution
Need(s) addressed, user segments	Control (Innovative-Users)	Control & Lock-out (Innovative-Users), then Adaptable (Innovative- + Frontier- Users)	Adaptable (Innovative- + Frontier-Users), Lock-out	Adaptable, possibly Low-cost (All Users)
Customer relationship & Channel	One-to-one services via the project	Club of sponsors who buy reputation & access to project control	Partially customized to automated services via an online platform	Standard services via an online platform
Key activities	FLOSS development	FLOSS project management (software, administration, legal aspects)		
Key resources	FLOSS developers	FLOSS developers, project managers, development platform	Brand	
Key partners	FLOSS project developers	FLOSS project, core developers	FLOSS project, core developers Module producers	
Cost structure	Human resources: developers	Human resources: developers & project managers Assets: Project development and management infrastructure		
Value capture	Billing for development hours	“Membership” fees	Certification services	Human resources: marketing Billing of standard assurance/assistance services
Software contribution from innovative-users				

The specific implementation of a FLOSS solution and the production of assured versions rely on some common resources, mainly related to the management of the dynamics of the project’s core, via a (strong) participation as core developer(s) (Linåker and Regnell, 2020). It is not surprising that an organization which relies more on the first model at the beginning of Phase 2 progressively develops the second model as the project grows and the users diversify (the history of AdaCore presented by Brosgol, 2019, –p. 19-20– also illustrates how a company paid to develop a FLOSS product for innovative users has become the editor and then publisher of the Ada language’s GNAT compiler and libraries). For these models, the owner of the brand, and of the intellectual property, associated with the FLOSS project benefits from the project’s reputation, signaling to customers that they are the go-to choice to provide (sell) guarantees and expertise on the generally eponymous FLOSS project (Schaarschmidt et al., 2015). Most of the time the editor takes the same name as the project. However, the two main capacities which were indistinguishable in Phase 1 (specifying needs and managing complex software projects) are increasingly dissociated in this phase, and an actor can choose to focus on one model only.

Modular architecture also opens up new avenues for models that are less dependent on the project’s core, focusing on developing or modifying specific modules.

Peripheral business models: consultant and module publisher. These peripheral business models are summarized in Table 8 at the end of the paragraph.

If the modules are developed in an open-source way, they can have the same advantages for a user as the core of the solution: no licensing costs, testing, compliance with standards, adaptation facilitated by opening the code. In this case, some companies can provide module adaptation for Innovative-users who chose to externalize this task or Frontier/No-skill users who need it. Providers can offer consulting and customization, combining expertise on their client’s business and on its specific needs, and technical expertise on modules. They can address “control” needs (fine-tuning, without lock-in), but benefit less from economies of scale, particularly from learning effects. We call them open-source “*business-solution consultants*”. The open-source ERP Odoo has a rich ecosystem of such consulting companies, such as ITransition. This can evolve towards the supply of a maintained technical capacity, if it succeeds in organizing the editing activities around the modules, and therefore

in selling the access to this maintained solution. These “*business-module publishers*” benefits from the fact that the core of the solution remains free, reducing the cost for the customer. Some sell the modules under classic private licenses. It is not an open-source business model, even if it contributes to the richness of the FLOSS project’s solutions. These offers do not respond to Control or Lock-out needs, but possibly to Adaptable needs. Cybrosys, which presents itself as "the Worlds Largest Odoo App Contributor Company", produces a mix of free and paid modules.

There is no need for these two models to develop expertise on the core of the application, thanks to the modular and open architecture. However, providers, especially business-module publishers, need to make sure that their modules remain compatible with the core, and to contribute to the core to do so.

We will summarize this by saying that, once again, the two main capacities needed to obtain a software solution lead to two main open-source business models. The first, addressing needs definitions and software adaptation, is mostly based on human skills and the sale of expertise time, with little economies of scale. The second, addressing software development, and selling (granted) versions of maintained technical solutions provide more room for economies of scale.

Table 8: Open-source peripheral business models in Phase 2: value addressed, captured and resources (each business model is presented in on column).

	Business Consultant	Business module publisher
Value proposition	Need specification / solution delivery services	Standard business- specific modules within a FLOSS solution
Need(s) addressed, user segments	Adaptable, any user (outsourcing or assistance) Adaptable, any user (outsourcing or assistance)	
Customer relationship & Channel	One-to-one services	Standard B2B services (module selling platform)
Key activities	Developing a human expertise on businesses and in their instantiation in code	
Key resources	Human: need specification	Human: need specification & software development
Key partners	None in the FLOSS project ecosystem	FLOSS project as a stable platform
Cost structure	Human resources: need specification	Human resources: developers and need specification Development infrastructure
Value capture	Billing for consulting hours	Selling assurance on modules

3.3.3. Phase 2 synthesis

The open-source models which have emerged in Phase 2 for the core of the solution diffuse toward the periphery, with the same mechanism of being paid to manage the software flow and adapt a solution, the modules, to clients' needs. The architecture of participation allows this, but also grants the FLOSS project's sustainability. By driving the newcomers' additional efforts to the periphery, the core remains manageable and its participants to retain the benefits of their participation for themselves. It also provides the project with the resources to expand the range of needs it addresses, and thus the clientele that can be targeted by the peripheral business models and the editor (e.g. more demand for the certified, assured versions, or for use assistance).

Finally, we want to point out that these business models are coherent the results of Zhang et al. (2021) on OpenStack, from the core-periphery contribution distribution to the open-source business models found and especially the link between their commercial offer and their involvement in the project, even if the scope of OpenStack makes that several editors models seem to cohabit and that each company's specific activity would need deeper analyses. However, it seems clear to us that their full-solution model reflects our publisher business model; their self-business-oriented model seems close to our business module publisher; and their specific sub-solution model regroups companies that propose an outsourcing solution. The business model of some companies in this ecosystem may propose business solution publishers, which we theorize is more common in the mature phase of a project, or Phase 3, which we will discuss in the next session. This is not surprising, as OpenStack is considered a mature, and even sometimes outdated solution²⁴.

3.4. Phase 3: Maturity and beyond

3.4.1. FLOSS projects: core maintenance and peripheral developments

A software project always needs technical updates, if only because of the constant evolution of the associated technologies (hardware and software). However, the number of contributors needed, as well as the intellectual appeal of the project, decrease as the project matures, as showed by Guimarães et al. (2013) and Joblin et al. (2023, Table 1, p. 14-15), among others. The projects

²⁴See the discussion of this in Jeffrey Burt's 2023 article for The Next Platform: "Reports Of OpenStack's Death Greatly Exaggerated" (read January 10 2025).

is also threatened by the natural tendency toward structural inertia and by an increasing difficulty to adapt. This may be due to technical decay that increases over time (Le et al., 2018) and/or, as previously mentioned, because of an overly rigid organization, like in traditional organizations (O'Mahony and Ferraro, 2007). Finally, a project can simply be abandoned by its core developers that do not wish to keep investing time in it, or because they consider it to be finished (Coelho and Valente, 2017; Ehls, 2017). Over a list of almost 2,000 popular GitHub projects Avelino et al. (2019) found that more 15% were abandoned by their maintainers²⁵.

Although there is less "vendor lock-in" with FLOSS, technological lock-in remains, as does the cost of maintaining an aging solution. Innovative-users will increasingly seek/initiate younger (FLOSS) projects, that they can more easily adapt (Control), with less technological lock-in (Lock-out)(See Lundell et al., 2017, on that aspect). Peripheral developments may continue for a while (new or updated modules), provided that the global solution, and its core, remain maintained; if the core maintainers leave it can be hard to find people to taking up the maintenance role. Less than half of the abandoned projects studied by Avelino et al. (2019) found new maintainers.

The project must organize this maintenance and its financing.

It is tempting for a private editor (publisher or consultant, but also user-editor) to stop maintaining the FLOSS version, as did the ERP publisher OpenBravo in 2010. This is to force users trapped in technological lock-in to pay for access (mainly the publisher), but mostly because community feedback and contributions are less frequent and thus less valuable compared to the cost of maintaining an architecture of participation (all editors). The decline in investment in community management accelerates the exodus of contributors, as stressed by Shaikh and Henfridsson (2017, in their discussion section). The abandonment of CentOS (for Community ENTERprise Operating System) Linux by RedHat, after the company stopped publishing the source code of its own Linux distribution, RedHat Enterprise Linux, for free, is one of the latest examples (See a summary of this by Nouredine, 2023).

The project's business model is no longer the user-as-innovator principle, in which value was created mainly by those who needed it (directly or indirectly),

²⁵Osdev.org maintains a list of 76 mostly FLOSS operating systems that were abandoned by their developers as of January 2025.

and by the coordination of this value creation (software flow management). Rather it is the organization of the financial flow between the users of the solution (core and modules) and the maintainers of the core, and increasingly of the modules (see Table 9). Open-source business models are able to organize such a financial flow.

Table 9: Business model for a FLOSS project in phase 3

Key partners Innovatives and open-source companies using the modules	Key activities Software core maintenance Modules coordination	Value proposition Maintenance of a stable solution which can be a support for adaptation to specific needs offers (via the modules)	Customer Relationships Download of the solution	Customer Segments Standard needs (Adaptable/Low-cost, all users)
	Key resources Project Maintainers, Brand		Channels Distribution platform	Specific needs addressed by specific modules: module producers (Adaptable, Frontier- or Non-skilled- users, represented by open-source companies)
Cost structure (time & money) Human resources: Maintenance time (reduced requirements compared to Phase 2) Assets: Software infrastructure established in Phase 2 must be maintained		Value capture (time & money) Module users fund core maintenance developers. Publishers and facility managers can also participate in maintenance to ensure the sustainability of the solution. (In both cases, directly to ensure compatibility, or through a foundation.)		

3.4.2. *Open-source business models: Maintaining a mature solution*

FLOSS editors. If the project remains FLOSS, it can be outsourced to a foundation at this stage, since, as explained in Phase 2, organizing such a financial flow is one of the core services it provides. If, to our best knowledge, the reasons why companies give money to projects have never been studied, we defend the idea that by donating to a project, actors are identified as sponsors, as explained on the operating system Linux Mint’s dedicated page which facilitates access to customers interested in peripheral services. Identifying and certifying distributors can also be a source of revenue for the editor.

The various outsourcing strategies than can be found in the software industry (Tadelis, 2007; Weeks and Feeny, 2008) can also be applied to FLOSS, from the most customized services to the most standard cloud services.

Some companies outsource part, or all of their infrastructure, but are seeking a solution tuned to their needs and maintained (between Adaptable and Control). FLOSS components can be part of this IT facility management service, with the mentioned advantages regarding flexibility and maintenance costs. This option may also reduce the risk of lock-in by offering a way out, albeit a theoretical one, in case of dissatisfaction with the supplier.

There are “*open-source outsourcing*” specialists (e.g. Smile, which presents itself as “the European leader in integration and open-source outsourcing solutions”). Mouakhar and Benkeltoum (2020) describe such a business model with their FACT’Open case (p. 25). Table 10 describes this model. Traditional outsourcing specialists have also integrated FLOSS in the technologies/software they follow (e.g. Infosys, which “offers an extensive range of services that help enterprises adopt open-source at scale, accelerate business innovation, [...] and contribute back to communities.”) Some of them, such as IBM with RedHat, have bought FLOSS specialists to do so. This model relies on economies of scale because the same IT solutions can be monitored and maintained for different clients, and on economies of scope because several software solutions are managed for the same client. The level of investment in the FLOSS project depends on its importance for the customers: relying on official versions, getting a little more involved in tracking bugs and fixing them, or, if it is core software for the infrastructure, having one or two developer-contributors to monitor the (weaker) software flow, in a much less costly way than in Phase 2.

Table 10: Open-source outsourcing business model

(Key partner: FLOSS project) Some FLOSS projects that are at the core of the infrastructure managed may be important to follow	Key activities FLOSS project assessment IT infrastructure management	Value proposition Maintenance services for the client's IT infrastructure	Customer Relationships Technical, service based	Customer Segments Lock-out (all users), even if risk of vendor lock-in
	Key resources IT/Software management (humans and infrastructure)		Channels One-to-one B2B services	
Cost structure (time & money) Human resource: developers and IT managers		Value capture (time & money) Billing for development/ maintenance/ management hours		

An open-source publisher can deploy a SaaS solution, for Frontier or even No-skill customers with Low-cost or Adaptable needs, based on the FLOSS software it controls (e.g. the WordPress platform; Gangadharan (2017) proposed a list of the existing Open Source Solutions for Cloud Computing). It is best positioned to do so when its name is associated with the project (brand), and because it owns the project management infrastructure. However, developing a SaaS solution requires new investments in the SaaS software and hardware platform, and specific human capacities to manage the platform (Bezemer and Zaidman, 2010). If this strategy succeeds, this “*open-source SaaS publisher*” model can benefit from a recurring flow of low unit subscriptions, but also from significant economies of scale thanks to the extreme standardization of the source code deployed and the machines managed.

Table 11: Open-source SaaS publisher business model

<p>(Key partner: FLOSS project) Peripheral modules and their developers may be important for the attractiveness of the solution</p>	<p>Key activities SaaS platform management Maintenance of the FLOSS solution</p> <p>Key resources Software management infrastructure, SaaS platform Brand</p>	<p>Value proposition Provision of a standard maintained hosted solution</p>	<p>Customer Relationships Standard B2B services</p> <p>Channels Online platform</p>	<p>Customer Segments Low-cost (All users)</p>
<p>Cost structure (time & money) Human resource: SaaS infrastructure (human and software)</p>		<p>Value capture (time & money) Standard pay-per-use solution</p>		

Drawing on the two models, the “*cloud platforms*” integrate FLOSS into their software libraries, with the same advantages and the same investment in the projects as outsourcing companies, and the same economies of scale as the SaaS. These platforms rely on FLOSS products to build their own infrastructure, and thus have the competences to follow FLOSS projects²⁶.

²⁶Wankhede et al. (2020) give an idea of the FLOSS solutions that are used by the three main cloud platforms, Microsoft Azure, Google cloud platform and Amazon EC2.

Table 12: "Open-source" Cloud business model

(Key) partner: FLOSS project Some FLOSS solutions may be important for the attractiveness of the platform	Key activities Cloud platform management	Value proposition Provision of a catalog of low cost maintained standard solution	Customer Relationships Standard services	Customer Segments Low-cost (All users)
	Key resources Cloud infrastructure, brand		Channels Online platform	
Cost structure (time & money) Mainly Cloud infrastructure		Value capture (time & money) Standard solution pay-per-use		

For these SaaS companies, the core competencies are much less about FLOSS project management than about SaaS platform management.

Peripheral business models. The transition into Phase 3 of the FLOSS modules may be delayed compared to the software core, and Phase 2 open-source models may prevail .

When modules become sufficiently stable, providers with strong business and coding expertise can develop a stand-alone solution that addresses a specific need. The competition will be based on business needs, rather than on the FLOSS aspects (even if using a FLOSS core may offer a price advantage), for these “*open-source business solution publishers*” (Table 13 below), that are exemplified by the GR’Open case in Mouakhar and Benkeltoum (2020). With a stable core, providers may choose to develop specific distributions, in order to capture all of the revenue from assurance services, or even from installation/adaptation to the customer’s IS, a classic verticalization strategy in the IS industry (Fink and Markovich, 2008). Coop-IT Easy, for instance, proposes Odoo-based solutions for the social economy. Providers may also attempt to make their module into a multi-platform solution (Askimet created an anti-spam extension for WordPress, and ported it to other content management systems such as Drupal).

Table 13: Open-source business solution publisher business model

(Key) partner: FLOSS project Floss projects as support for the modules developed	Key activities Developing a human expertise on businesses and in their instantiation in code	Value proposition Stand-alone business-specific solutions	Customer Relationships Standard B2B services (complete solution)	Customer Segments Adaptable, Frontier/Non-skilled users
	Key resources Human (software development)		Channels Online platform	
Cost structure (time & money) Human resources (developers and need specification) & Development infrastructure		Value capture (time & money) Selling assurance on a complete business solution		

4. Discussion and Conclusion

FLOSS projects have long been studied by software engineering research, as they provide insights into the organization of a group development process (the reference article on this matter seems to be Crowston et al., 2006), and into how companies can evaluate a FLOSS project/product (for a summary of such metrics and tools, see Jansen, 2014). Today, the analysis of FLOSS organizations has shifted to understanding how the project organization can be reproduced internally to foster innovation (Stol et al., 2011, 2014), and more recently, how it impacts the business of adopters (Buchner and Riehle, 2023). The concept of continuous software engineering (Fitzgerald and Stol, 2017), i.e. the idea that a software project needs to be understood as the continuous integration of software development, its operational deployment and business strategy, emphasizes the need to rethink the fundamentals of how a FLOSS project works.

This is because understanding why a company chooses a FLOSS product, and why open source companies are needed in the adoption of such products, remains something to be investigated, as shown by the recent survey by Li et al. (2022). There are probably several reasons for this lack of integrated analysis.

One is that most of these studies focus on mature FLOSS projects, despite the knowledge that FLOSS projects are heterogeneous (Krishnamurthy, 2002; Healy and Schussman, 2003), in terms of size, ambition, but also stage of development (Koch, 2008; Jullien et al., 2019; Liao et al., 2019; Setia et al., 2020). We have argued that this limits both our understanding of how these projects work, but also why and how actors, especially companies, invest in them, and what are the open source business models that benefit and support FLOSS projects.

Therefore, this article has attempted to inform these concepts by discussing the key questions they raise about the organization of FLOSS projects and the participation of companies throughout the life of the project: how does a FLOSS project organize a value network of participants, over its different development phases, and how does this affect the business models (plural) of a specific type of participants, namely open source companies.

To do so, this article has drawn on the theoretical proposition of Morgan et al. (2013) to see FLOSS projects as "value networks", i.e. an "organization of the cooperation of different actors in the production and capture of value that

they use to satisfy their needs". For each phase, we have therefore examined what value this network produces, for whom, and using which resources, i.e. the project's business model. This led to a more precise definition of what open source companies propose in order to complement the project and with which resources, one of them being their level of involvement in the FLOSS project.

We summarize here the dual dynamics that characterize FLOSS projects (research question 1) and open-source business models (research question 2), before suggesting implications for practitioners and concluding with research needs.

4.1. FLOSS projects, open-source business models

FLOSS projects go through phases that are characterized by the way the collaboration is structured in terms of governance, production process, and solution provided (value captured and value created by the project). As previously explained by Morgan and Finnegan (2014), benefiting from the solution implies efforts to adapt the solution to one's needs, but also by participating in the production of code and in the governance of the project to adapt the solution to one's needs. We have completed this analysis by showing that the needs met and the efforts made vary according to the phase of the project, and, as a consequence the open-source models are also phase dependent.

4.1.1. RQ1: the phases of a FLOSS project

Most projects remain small, allowing a group of one or a few actors to collaborate on a solution which mainly serves themselves (FLOSS Phase 1). Some, because of their scope, and their success, become structured technical projects which organize different levels of participation and allow for several open-source business models (FLOSS Phase 2). The architecture of participation is also an architecture of selection, giving those who have access a competitive advantage in providing the services mentioned. Phase 2 can last years, even decades (e.g. Linux). The choice of model for coordinating value creation (the project's editor), between private companies, foundations, or platforms depends on who is influential at the project's inception. Finally, when the solution matures, the value proposition of the FLOSS project, which consists in coordinating actors willing to contribute their needs and ensuring the code remains open, loses some of its attractiveness. Table 14 summarizes the FLOSS business models by phase.

Table 14: The different business models of a FLOSS project by phase.

	Phase 1	Phase 2		Phase 3
		core development project	peripheral module projects	
Value Creation				
Value Proposition	Controlled & personalized software solution	Modular open software solution	Adaptation to specific needs of a stable solution (Control/Adaptable)	Maintenance of a stable solution which can be tailored to specific needs (via the modules)
Need(s) addressed, user segments	Control (Innovative-users)	Control, lock-out (Innovative- or Frontier-users represented by open-source companies)	Control (Innovative-users), Adaptable (Innovative- & Frontier- users, represented by open-source companies)	Standard needs (Adaptable/Low-cost, all users) Specific needs addressed by specific modules: module producers (Adaptable, Frontier- or Non-skilled-users, represented by open-source companies)
User relationship & Channel	The users are the producers, interaction through the project's platform	The users are the producers (directly or through open-source companies). Interaction through the development platform	Download of the solution. Interaction through a distribution platform	Download of the solution. Interaction through a distribution platform
Resources				
Key activities	Software development	Software development	Module development & coordination	Software core maintenance Module coordination
Key Resources (assets & capacities)	Development platform Project manager	Development platform (architecture of participation) Project manager Project editor	Module developers Brand	Project Maintainers Brand
Key partners	Innovative-users	Innovative-users & open-source companies for both the core & the modules (not always the same users)		Innovative-users & open-source companies using the modules

	Phase 1	Phase 2		Phase 3
		core development project	peripheral module projects	
Value Flow				
“Cost structure” (time & money)	Development time (development & project management)	Human resource: Developers time (development & project management). Assets: software infrastructure (free online solution like GitLab can be used)		Human resources: maintenance time (reduced requirements compared to Phase 2). Assets: the infrastructure built in Phase 2 must be maintained
Value capture (time & money)	Developer time	Core developer time. Contributions from module users/ open-source companies to ensure compatibility	Module developers	Module users, publishers, & facility managers fund core maintenance developers (directly to ensure compatibility, or through a foundation).
Examples (from the academic environment to complete those presented in the text)				
	PPanGGOLiN is "a software suite used to create and manipulate prokaryotic pangenomes from a set of either genomic DNA sequences or provided genome annotations" Gautreau et al. (2020). The number of contributors remains small (12) but goes beyond the original team that signed the scientific paper presenting it. PPanGGOLiN has received the ‘promising’ award of the French Ministry of Higher Education and Research’s 2023 Free Research Software Awards.	The Coq interactive theorem prover, which has been in development for 35 years, is a good example of a long-running project that still generates a lot of peripheral (module) development, but is also still active in its core development. The history of the project is presented in its Wikipedia page.		The Scala language project, which present itself as "a mature open-source project" is maintained by a core team of developers at EPFL, that does not accept direct contributions to the core project (no open git repository). The contribution process and the governance are very formalized and hierachical. There are still a lot of peripheral modules being developed.

Alongside the project's life the efforts and investments made by the project and its contributors evolve from the coordination of experts to the industrialization of a software production, and from the collaborative development of a core solution, a user-as-innovator open collaboration, to the coordination of different peripheral contributions addressing different needs based on a core technology, something closer to an (open) innovation platform.

The more dynamic a FLOSS project, the stricter this selection process, but also the more business opportunities for open-source companies (RQ2).

4.1.2. RQ2: open-source business models are FLOSS phase-dependent

Open-source models are models of service, the commercial provision of what Jullien and Zimmermann (2009) called "3A services": 1) assistance with needs specification, 2) adaption of software to the organization, and the organization to the software, both of which are based on human capacities 3) access to a maintained/assured technical capacity, or "assurance", based on the mastery of a software factory, which is a combination of technical tools and human skills.

This is not really new in terms of value creation, as these are the two main capacities we have been discussing since the introduction. However, when the solution is developed by a FLOSS project, open-source companies have to invest time and effort in contributing to it in order to be able to provide the services sold: software project management <-> help and assurance on the core software flow; module development, selection or customization <-> user requirements specification and instantiation (Schaarschmidt, 2023).

The capacity of open-source organizations (communities, companies or foundations) to capture a part of the value created is reinforced by the control of dynamic immaterial assets: the brand (if they own it and if the project is recognized and is dynamic) but also databases (of bugs, for example, which must be constantly updated), which are kept secret.

Our work concurs with Lin and Maruping (2022) in that open-source business models depend on and co-evolve with the FLOSS project, its maturity, but also its scope, which will influence the level of investment an open-source company must maintain in the FLOSS project (value contribution to the project), and the value it can capture. This has been illustrated by the different business models we identified, that are summarized in Table 15, for the editor models, and Table 16.

Table 15: Open-source Editor business models: value addressed, captured and resources.

		Open-source Editor business models			SaaS publisher	
		Customization	Foundation	Ecosystem manager		Publisher
FLOSS Phases:		1,2	2,3	2,3	3	
Value creation						
Value proposition	details	Custom development within a FLOSS project	Participant-Neutral Management of a FLOSS project	Management & certification of FLOSS actors/modules	Provision of a standard maintained FLOSS solution	Provision of a standard maintained hosted solution
	software-stock access	--*	--	-	+	++
	assurance (access to software-flow)	+	+	++	++	+
	assistance	+	-	+	+	-
	adaptation	++	--	--	--	--
Need(s) addressed, user segments		Control (Innovative-Users)	Control & Lock-out (Innovative-Users), then Adaptable (Innovative-Users)	Adaptable (Innovative-Users), Frontier-Users), Lock-out	Adaptable, possibly Low-cost (All Users)	Low-cost (All Users)
Customer relationship & Channel		One-to-one B2B services via the project	Club of sponsors who buy reputation & access to project control	Partially customized to automated services	Standard B2B services via an online platform	Standard B2B services via an online platform

* Legend: - - (very unimportant), - (unimportant), + (important), ++ (very important) for the open-source business model.

		Customization		Open-source Editor business models		
				Foundation	Ecosystem manager	Publisher
FLOSS Phases:		1,2	2,3	2,3	2,3	3
Resources						
Key Activities	To build the value proposition:	FLOSS development	FLOSS project management (software, administration, legal aspects)			
	FLOSS project participation:	Central	Core/ main contributor, project manager	Core/ main contributor, project manager. May develops modules	SaaS platform management	
Key resources	Human (need specification)	++	+	-	-	--
	Human (software development)	++	++	++	++	+
	Knowledge based IT (bugs, needs database)	+	++	++	++	+
	Software management infrastructure	+	++	++	+/++	++
	Brand, other IP such as logos	N/A	++	++	++	++
Partner: FLOSS project	Core project evolutions	++	++	+	+	-
	Peripheral evolutions	N/A	++	++	++	+

* Legend: - - (very unimportant), - (unimportant), + (important), ++ (very important) for the open-source business model.

		Open-source Editor business models				
		Customization	Foundation	Ecosystem manager	Publisher	SaaS publisher
FLOSS Phases:		1,2	2,3	2,3	2,3	3
Value Flow						
Cost structure	Human resources (developers)	Human resources (developers & admin.) & Development infra.	Human resources (developers) & Development infra.	Human resources (developers) & Development infra.	Human resources (developers & marketing) & Development & diffusion infra.	Mainly SaaS infra. (human and software)
Value capture	Billing for development hours	“Membership” fees		Certification services	Billing of standard assurance/ assistance services	Standard pay-per-use solution
Examples						
	Ideas on Board provides “Software consultancy for Multimedia in Linux”	In addition to the foundations cited in the text, The Eclipse Foundation provides “a business-friendly environment for open-source software collaboration and innovation.”	The ERP Odoo company and its certified partners or the Linux Foundation with Kubernetes and its certificates		Canonical offers such a solution for the Linux based operating System Ubuntu	GitHub or GitLab, two providers of SaaS solutions for software development management. WordPress is another example.

Table 16: Other open-source business models (than editor): value addressed, captured and resources.

		FLOSS solution providers		Peripheral business model		
		Outsourcing	Cloud platforms	Business consultant	Business module publisher	Business solution publisher
FLOSS Phases:		3	3	2, 3	2, 3	3
Value creation						
Value proposition	details	Maintenance service for the client's IT infrastructure	Provision of a catalog of low cost maintained standard solutions	Need specification / solution delivery services	Standard business-specific modules within a FLOSS solution	Stand-alone business-specific solutions
	software-stock access	-	+ +	+ / -	+	+
	assurance	+	+	- -	+	+
	assistance	+	- -	++	+	++
Need(s) addressed, user segments	adaptation	-	- -	++	+	+
		Lock-out (All Users), even if risk of vendor lock-in	Low-cost (All Users)	Adaptable, any user (outsourcing or assistance)	Adaptable, any user (outsourcing or assistance)	Adaptable, Frontier/Non-skilled users
Customer relationship & Channel		One-to-one B2B services	Standard B2B services via an online platform	One-to-one B2B services	Standard B2B services (module selling platform)	Standard B2B services (complete solution) via an online platform

* Legend: - - (very unimportant), - (unimportant), + (important), + + (very important) for the open-source business model.

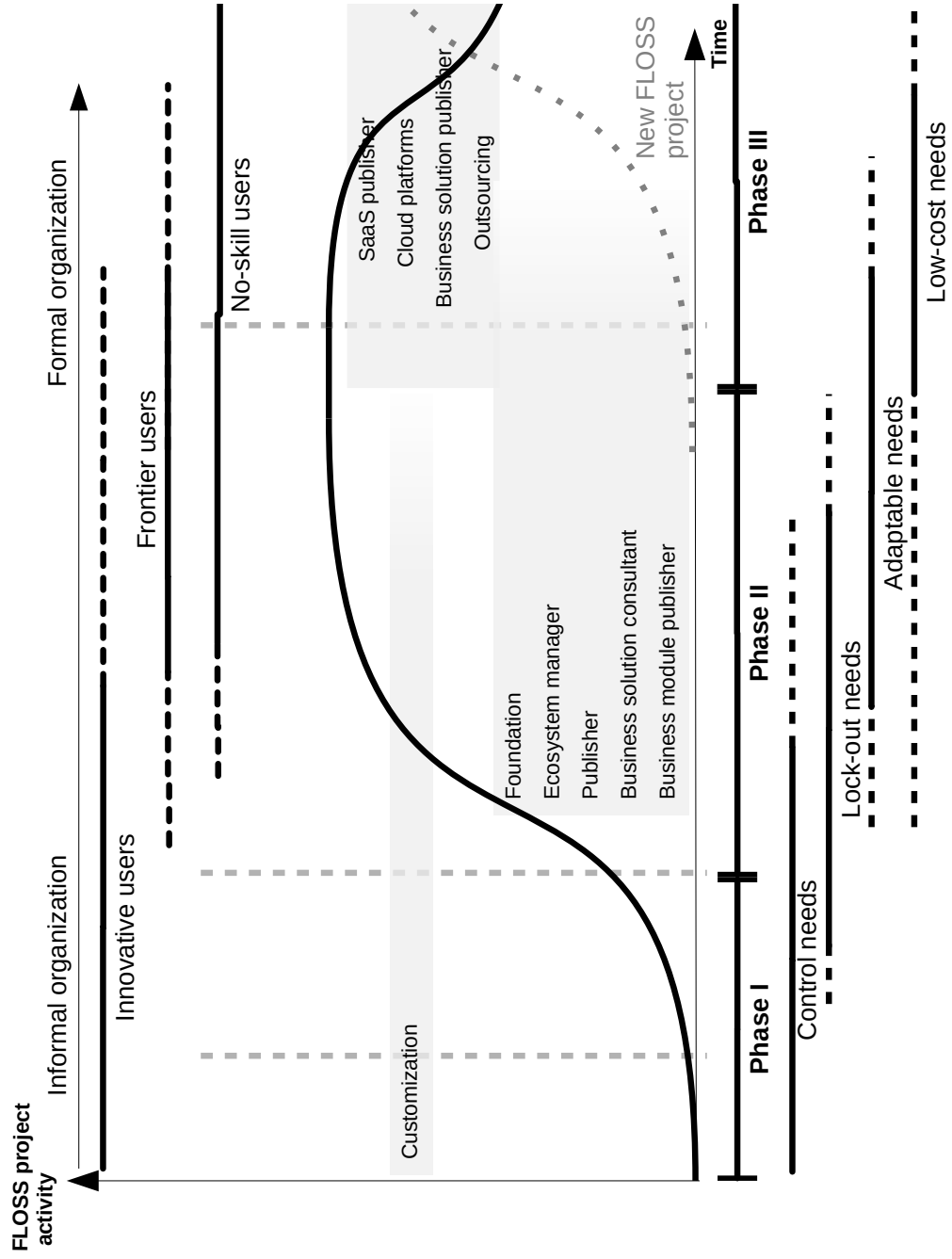
		FLOSS solution providers		Peripheral business model		
		Outsourcing	Cloud platforms	Business consultant	Business module publisher	Business solution publisher
FLOSS Phases:		3	3	2, 3	2, 3	3
Resources						
Key Activities	To build the value proposition:	FLOSS project assessment; IT infrastructure mgmt	Cloud platform management	Developing a human expertise on businesses	Developing a human expertise on businesses and in their instantiation in code	
	FLOSS project participation:	Peripheral participation	Very peripheral participation	No participation	Active participation in own module dvpt; periph. parti. in core dvpt	Own modules, periph. part. in core dvpt
Key resources	Human (need specification)	+	--	++	+	+
	Human (software development)	+	+	-	++	++
	Knowledge based IT (bugs, needs database)	+	+	-	+	+
	Software management infrastructure	++	+	-	+	+
	Brand, other IP such as logos	-	++	+/-	+	+
Partner: FLOSS project	Core project evolutions	+ / -	-	N/A	+	- / +
	Peripheral evolutions	+ / -	-	N/A	++	+

* Legend: -- (very unimportant), - (unimportant), + (important), ++ (very important) for the open-source business model.

	FLOSS solution providers		Peripheral business model		
	Outsourcing	Cloud platforms	Business consultant	Business module publisher	Business solution publisher
FLOSS Phases:	3	3	2, 3	2, 3	3
Value Flow					
Cost structure	Human resources (developers and IT managers)	Mainly Cloud infrastructure	Human resources (need specification)	Human resources (developers and need specification) & Development infrastructure	
Value capture	Billing for development/ maintenance/ management hours	Standard solution pay-per-use	Billing for consulting hours	Selling assurance on modules	Selling assurance on a complete business solution
Examples					
	The Smile company is “the European leader in integration and open-source outsourcing solutions”	AWS, Google Cloud, OVH-Cloud, Azure...	Open-source ERP Odoo consulting companies such as ITransition	Cybrosys, in the Odoo ecosystem	Coop-IT Easy's Odoo based solutions (vertical integration). Askimet anti-spam extension (multi-platform solution)

The articulation between the maturity of the FLOSS projects and the needs in each phase is summarized in Figure 3.

Figure 3: Life cycle and economic model of a FLOSS project



There is a form of path dependency between open-source business models, as there is a form of path dependency in the evolution of participants' roles: it is easier for a key project initiator to become a core developer than a newcomer. However, this is always a contestable position, for two reasons: if a player is seen as too invasive by the Innovative-users, if they feel their needs are not being respected, they may "fork" the project²⁷ or create a new one; second, the key competencies are not exactly the same at each stage. An actor must have the resources (finance, skills) to adapt.

Finally, at each stage, new players that already possess business capacities can capture part of the value by getting involved in the development, thus diminishing the incumbents' competitive advantage. This is not necessarily a bad thing for the project and its dynamics, but can explain the instability of the private stakeholders gravitating around it.

This also means that before choosing a FLOSS solution, or an open-source company, their models are to be evaluated.

4.2. Lessons for practitioners

4.2.1. Users: selecting a FLOSS project and an open-source company

For users, the choice of FLOSS solutions is governed by the same type of process as any software solution: defining the user's needs and resources, in terms of skills (Innovation, Frontier, or Naive user), as well as effort (time, money), but also assessing the solidity and sustainability of the collaborative project in terms of product and process quality, as well as the diversity of actors, as analyzed by Shaikh and Levina (2019). The fact that code has been released under an open license by a single person is not enough to qualify a piece of code as a FLOSS solution.

The four quadrants we have presented show different competitive advantages of FLOSS. In terms of control needs, a FLOSS project allows Innovative-users to share the cooperative development of a solution that is better adjusted to their needs, cheaper than an in-house project, and whose upgradability can be controlled over time. Other users delegate this control to providers, even if a FLOSS solution may facilitate the switch from one provider to another. Using a FLOSS solution may be interesting for lock-out needs, because it ensures better long-term compliance with standards, providing that the solution is

²⁷Forking means taking the source code and using it to create a new FLOSS project. For a historical analysis of a fork, see Gamalielsson and Lundell (2014).

dynamically developed by several actors. This can be evaluated by Innovative- and Frontier-users. For Adaptable demands, a FLOSS solution may reduce the cost of adaptations, as investment can be focused on the modules while the core solution is free of charge. Yet, once again, apart from Innovative-users, these adaptations are to be contracted to providers, and the provision of a good solution depends as much on the (FLOSS) solution as on the relationship between the provider and the client. Finally, in terms of Low-cost-related needs, an advantage can be found in FLOSS solutions: they are license-free. However, even a solution addressing these demands may face bugs or need technical evolution and is never cost-free.

For any use, a good FLOSS solution must be maintained, by more than one actor, or the risks are as high as using a free-of-charge private solution, for a gain (the license cost), which may not be that significant, as illustrated by the study of Orucevic-Alagic and Höst (2014) or the Android project. Table 17 details the benefits of FLOSS for each demand and user.

Table 17: Users' interest for FLOSS according to their need and their skills

Needs: Skills:		Low-cost	Lock-out	Adaptable	Control
		Assurance (-/+)* Assis- tance (-) Adaptation (--)	Assurance (+/+++)* Assis- tance (+) Adaptation (-)	Assurance (+) Assis- tance (+) Adaptation (++)	Assurance (++) Assis- tance (+/-) Adaptation (++)
Non- skilled user	Capacities to "choose" a solution	Information through price only.	Information through price, possibly online ratings; dif- ficulty in evaluating the lock-in.	Outsource, information through support in expressing needs, little control over the quality of the solution developed, difficulty in establishing an effective service relationship. Discussion about business needs, little about technical needs: trust in provider essential.	Looking for support in expressing needs, little control over the quality of the solution developed, difficulty in establishing an effective service relationship. Discussion about business needs, little about technical needs: trust in provider essential.
	FLOSS advan- tages (com- pared to classic private software)	Mature and well-known FLOSS solutions may be interesting as "fee-free" and easily accessible solutions.	The presentation of the software as "free" software can be a positive signal, but less so than the assurances of longevity (brand).	FLOSS can be chosen by the providers, if it makes sense for them, but not by the users. It can allow the service provider to reduce the total cost of the solution (savings on licenses, easier customization).	FLOSS can be chosen by the providers, if it makes sense for them, but not by the users. It can allow the service provider to reduce the total cost of the solution (savings on licenses, easier customization).

* Legend: - (very unimportant), - (unimportant), + (important), ++ (very important) for the need to be addressed by the software solution.

Needs: Skills:		Low-cost	Lock-out	Adaptable	Control
		Assurance (-/+)* Assis- tance (-) Adaptation (--)	Assurance (+/+++)* Assis- tance (+) Adaptation (-)	Assurance (+) Assistance (+) Adaptation (++)	Assurance (++) Assistance (+/-) Adaptation (+++)
Frontier user	Capacities to "choose" a solution	Information through price, but also on the ability to assess the adequacy to the needs	Capacity to test the prod- uct, possible exploration of the license	Capacity to test the product and to provide feedback on needs. Outsource, difficulty in evaluating the technical quality of the developed solution. Trust in the technical provider	Capacity to test the product and to provide feedback on needs. Outsource, difficulty in evaluating the technical quality of the solution and its scalability or durability.
	FLOSS advan- tages (com- pared to classic private software)	Mature and well-known FLOSS solutions may be interesting as "free" and easily accessible solutions	FLOSS can be viewed as Assurance against long-term vendor lock-in.	FLOSS can allow the solution to be tested at a lower cost than a classic solution. It can facilitate the discus- sion with the provider about the required adaptations.	It may facilitate the discus- sion with the provider on the required adaptations. It can also be viewed as As- surance against long-term vendor lock-in.

* Legend: - (very unimportant), - (unimportant), + (important), ++ (very important) for the need to be addressed by the software solution.

Needs: Skills:		Low-cost	Lock-out	Adaptable	Control
		Assurance (-/+)* Assis- tance (-) Adaptation (--)	Assurance (+/+++)* Assis- tance (+) Adaptation (-)	Assurance (+) Assistance (+) Adaptation (++)	Assurance (++) Assistance (+/-) Adaptation (++)
Innova- tive user	Capacities to "choose" a solution	Information through price; ability to assess the ade- quacy to the needs	Possibility to test the prod- uct both technically and in terms of meeting the needs. But costly activities	Possibility to test the product both technically and in terms of meeting the needs. Choice between doing or making, ability to express needs and control imple- mentation, in the short and long term. Investment in evaluation and production according to the criticality of the project.	
	FLOSS advan- tages (com- pared to classic private software)	FLOSS interesting as "free" software, easy to access and test (Phase 2-3 solutions, such as Firefox)	FLOSS can facilitate the tests (needs and techniques), decreasing their cost. Dynamic FLOSS projects with open governance can provide long-term protec- tion against vendor lock-in.	FLOSS solution potentially interesting because it is adaptable to needs; open project dimension important (who controls the project, is it possible to contribute the required adaptations). Dynamic, well-structured FLOSS projects can provide greater value by allowing for better adaptation to needs.	New or dynamic, well- structured, and open- governance FLOSS projects, that can bring greater value by allowing a better adap- tation to the needs in the short/ long term

* Legend: - (very unimportant), - (unimportant), + (important), ++ (very important) for the need to be addressed by the software solution.

4.2.2. Building an open-source business model

Companies willing to develop an open-source business model must understand that if the value created is the same as for classic models, its realization deals with specific capacities. To be able to assess a FLOSS solution means understanding its existing components and its evolution. To be able to adapt and assure a FLOSS solution means contributing to it. The level of contribution depends on what is provided (adaptation/assurance of the core, or adaptation/assurance of modules). Somehow, as the open-source companies act as a substitute for their Innovative-clients or in the name of their Frontier- or No-skill users, they have to develop the same participating capacities as an Innovative-user and find in the FLOSS project the same possibilities to address needs as Innovative-users. The Innovative-user in Table 17 can be an open-source company.

4.3. Conclusion: limits and perspectives

Research has given much of its attention to Phase 2 FLOSS projects, as illustrated by the guidelines/methods for assessing the health of FLOSS projects (for a synthesis of such metrics and tools, see Jansen, 2014). This is understandable, since it is here that the technical, management questions, as the open source models, are the most numerous, as also illustrated by this work (Phase 2 is the longest subsection). However, it is equally important to better understand the management of Phase 1 (how to identify a promising project) and Phase 3 (does a mature solution still offer FLOSS advantages?) The best governance structure for each phase of a FLOSS project, as well as how to manage the transition from one phase to another, also needs further investigation.

Open-source companies can also participate in several FLOSS projects and deploy different business models depending on the project. This is easier as the key resources are similar – the roles of project publisher and business-module publisher, for example, call upon the same types of skills (management of a maintained technical capacity) – or meet complementary customer needs.

This explains why research and practice have found it difficult to understand companies' open-source business models, and more specifically the link between participation in projects and value capture. We hope this work on the foundations of the FLOSS/Open-source business models helps elucidate these links, even if more research is still needed on some points.

In particular, the role of foundations as neutral actors coordinating competi-

tors, their governance, and the reasons why an editor decides to outsource its project to them, as why some actors sponsor some FLOSS projects by giving money to them warrant further study. This is also the case for the 3A services implemented in the peripheral open-source models, that, as in the computer industry in general, account for the largest share of business activity (again, see Markets and Markets’ “Open Source Services Market” 2021 report). Therefore, we advocate that the software engineering community improves FLOSS evaluation tools by taking into account the maturity of the solution (again, phases 1 and 3 are understudied and modeled), but also the strategic needs of the adopters.

References

- Ågerfalk, P. J. and Fitzgerald, B. T. (2007), ‘Outsourcing to an unknown workforce: Exploring opensourcing as a global sourcing strategy’, *MIS Quarterly* **32**(2), 385–409.
- Ait, A., Izquierdo, J. L. C. and Cabot, J. (2022), An empirical study on the survival rate of github projects, *in* ‘Proceedings of the 19th International Conference on Mining Software Repositories’, pp. 365–375.
- Almeida, F., Oliveira, J. and Cruz, J. (2011), ‘Open standards and open source: enabling interoperability’, *International Journal of Software Engineering & Applications (IJSEA)* **2**(1), 1–11.
- Amrit, C. and Van Hillegersberg, J. (2010), ‘Exploring the impact of socio-technical core-periphery structures in open source software development’, *journal of information technology* **25**(2), 216–229.
- Arthur, W. B. (1994), *Increasing returns and Path dependence in the Economy*, University of Michigan Press.
- Avelino, G., Constantinou, E., Valente, M. T. and Serebrenik, A. (2019), On the abandonment and survival of open source projects: An empirical investigation, *in* ‘ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)’, IEEE, pp. 1–12.
- Baden-Fuller, C. and Morgan, M. S. (2010), ‘Business models as models’, *Long range planning* **43**(2-3), 156–171.

- Baldwin, C. Y. and Clark, K. B. (2000), *Design rules: The power of modularity*, Vol. 1, MIT press.
- Baldwin, C. Y. and Clark, K. B. (2003), ‘The architecture of cooperation: How code architecture mitigates free riding in the open source development model’, *Harvard Business School* **43**.
- Baldwin, C. Y. and Clark, K. B. (2006), ‘The architecture of participation: Does code architecture mitigate free riding in the open source development model?’, *Management science* **52**(7), 1116–1127.
- Bennett, K., Layzell, P., Budgen, D., Brereton, P., Macaulay, L. and Munro, M. (2000), Service-based software: the future for flexible software, *in* ‘Proceedings Seventh Asia-Pacific Software Engineering Conference’, IEEE, pp. 214–221.
- Bessen, J. (2006), Open source software: Free provision of complex public goods, *in* ‘The economics of open source software development’, Elsevier, pp. 57–81.
- Bessen, J. E. and Maskin, E. S. (2009), ‘Sequential innovation, patents, and imitation’, *The RAND Journal of Economics* **40**(4), 611–635.
- Bezemer, C.-P. and Zaidman, A. (2010), Multi-tenant saas applications: maintenance dream or nightmare?, *in* ‘Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)’, ACM, pp. 88–92.
- Bogart, C., Kästner, C., Herbsleb, J. and Thung, F. (2021), ‘When and how to make breaking changes: Policies and practices in 18 open source software ecosystems’, *ACM Transactions on Software Engineering and Methodology (TOSEM)* **30**(4), 1–56.
- Bonaccorsi, A. and Rossi, C. (2003), ‘Why open source software can succeed’, *Research Policy* **32**(7), 1243–1258.
- Brosgol, B. M. (2019), ‘How to Succeed in the Software Business While Giving Away the Source Code: The AdaCore Experience’, *IEEE Software* **36**(6), 17–22.

- Buchner, S. and Riehle, D. (2023), ‘The business impact of inner source and how to quantify it’, *ACM Computing Surveys* **56**(2), 1–27.
- Butler, S., Gamalielsson, J., Lundell, B., Brax, C., Mattsson, A., Gustavsson, T., Feist, J., Kvarnström, B. and Lönroth, E. (2022), ‘Considerations and challenges for the adoption of open source components in software-intensive businesses’, *Journal of Systems and Software* **186**, 111152.
- Campbell-Kelly, M. and Garcia-Swartz, D. D. (2015), *From mainframes to smartphones: A history of the international computer industry*, Harvard University Press.
- Chakraborti, M., Atkisson, C., Stănciulescu, Ș., Filkov, V. and Frey, S. (2024), Do we run how we say we run? formalization and practice of governance in oss communities, in ‘Proceedings of the CHI Conference on Human Factors in Computing Systems’, pp. 1–26.
- Chesbrough, H. (2023), Measuring the economic value of open source, Technical report, The Linux Foundation.
- Chesbrough, H., Lettl, C. and Ritter, T. (2018), ‘Value creation and value capture in open innovation’, *Journal of Product Innovation Management* **35**(6), 930–938.
- Chesbrough, H. W. and Appleyard, M. M. (2007), ‘Open innovation and strategy’, *California management review* **50**(1), 57–76.
- Coelho, J. and Valente, M. T. (2017), Why modern open source projects fail, in ‘Proceedings of the 2017 11th Joint meeting on foundations of software engineering’, ACM, pp. 186–196.
- Cohen, W. M. and Levinthal, D. A. (1989), ‘Innovation and learning: The two faces of r&d’, *Economic Journal* **99**, 569–596.
- Crowston, K. and Howison, J. (2005), ‘The social structure of Free and Open Source Software development’, *First Monday* **10**(2).
URL: <http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/1207/1127>
- Crowston, K., Howison, J. and Annabi, H. (2006), ‘Information system Success in Free and Open Source Software Development: Theory and Measures’, *Software Process Improvement and Practice* **11**, 123–148.

- Crowston, K. and Scozzi, B. (2002), ‘Open source software projects as virtual organisations: competency rallying for software development’, *IEEE Proceedings-Software* **149**(1), 3–17.
- Crowston, K., Wei, K., Howison, J. and Wiggins, A. (2008), ‘Free/libre open source software development: What we know and what we do not know’, *ACM Computing Surveys* **44**.
- Cusumano, M. (2004), *The Business of software: what every manager, programmer, and entrepreneur must know to thrive and survive in good times and bad*, Free Press, New York.
- Cusumano, M. A. (2008), ‘The changing software business: Moving from products to services’, *Computer* **41**(1), 20–27.
- Dahlander, L. and Wallin, M. W. (2006), ‘A man on the inside: Unlocking communities as complementary assets’, *Research Policy* **35**, 1243–1259.
- Daniel, S., Agarwal, R. and Stewart, K. J. (2013), ‘The effects of diversity in global, distributed collectives: A study of open source project success’, *Information Systems Research* **24**(2), 312–333.
- Dasgupta, P. and David, P. A. (1994), ‘Toward a new economics of science’, *Research Policy* **23**(5), 487–521.
- De Laat, P. B. (2005), ‘Copyright or copyleft?: An analysis of property regimes for software development’, *Research Policy* **34**(10), 1511–1532.
- Demil, B. and Lecocq, X. (2010), ‘Business model evolution: in search of dynamic consistency’, *Long range planning* **43**(2-3), 227–246.
- Duparc, E., Möller, F., Jussen, I., Stachon, M., Algac, S. and Otto, B. (2022), ‘Archetypes of open-source business models’, *Electronic Markets* **32**(2), 727–745.
- Duvall, P., Matyas, S. and Glover, A. (2007), *Continuous Integration: Improving Software Quality and Reducing Risk*, Addison-Wesley Signature Series, Pearson Education.
- Economides, N. (2001), ‘The microsoft antitrust case’, *Journal of Industry, Competition and Trade* **1**, 7–39.

- Ehls, D. (2017), Open source project collapse - sources and patterns of failure, *in* ‘Proceedings of the 50th Hawaii International Conference on System Sciences’.
- Ethiraj, S. K., Kale, P., Krishnan, M. S. and Singh, J. V. (2005), ‘Where do capabilities come from and how do they matter? a study in the software services industry’, *Strategic Management Journal* **26**(1), 25–45.
- Faria, H. and Belfo, F. P. (2019), ‘Strategic quadrant for companies implementing open source erp systems-cases of odoo erp implementers in portugal’, *19 Conferência da Associação Portuguesa de Sistemas de Informação (CAPSI’2019)* .
- Farrell, J. (1995), ‘Arguments for weaker intellectual property protection in network industries’, *StandardView* **3**(2), 46–49.
- Ferraz, I. N. and Santos, C. D. d. (2021), ‘Organization of free and open source software projects: In-between the community and traditional governance’, *BBR. Brazilian Business Review* **18**(3), 334–352.
- Fink, L. and Markovich, S. (2008), ‘Generic verticalization strategies in enterprise system markets: An exploratory framework’, *Journal of Information Technology* **23**(4), 281–296.
- Fitzgerald, B. (2006), ‘The transformation of open source software’, *MIS Quarterly* **30**(3), 587–598.
- Fitzgerald, B. and Stol, K.-J. (2017), ‘Continuous software engineering: A roadmap and agenda’, *Journal of Systems and Software* **123**, 176–189.
- Franke, N. and von Hippel, E. (2003), ‘Satisfying heterogeneous user needs via innovation toolkits: the case of Apache security software’, *Research policy* **32**(7), 1199–1215.
- Gamalielsson, J. and Lundell, B. (2014), ‘Sustainability of open source software communities beyond a fork: How and why has the libreoffice project evolved?’, *Journal of Systems and Software* **89**, 128–145.
- Gambal, M.-J., Asatiani, A. and Kotlarsky, J. (2022), ‘Strategic innovation through outsourcing—a theoretical review’, *The Journal of Strategic Information Systems* **31**(2), 101718.

- Gangadharan, G. (2017), ‘Open source solutions for cloud computing’, *Computer* **50**(01), 66–70.
- Gautreau, G., Bazin, A., Gachet, M., Planel, R., Burlot, L., Dubois, M., Perrin, A., Médigue, C., Calteau, A., Cruveiller, S. et al. (2020), ‘Ppangolin: depicting microbial diversity via a partitioned pangenome graph’, *PLoS computational biology* **16**(3), e1007732.
- Germonprez, M., Kendall, J. E., Kendall, K. E., Mathiassen, L., Young, B. and Warner, B. (2017), ‘A theory of responsive design: A field study of corporate engagement with open source communities’, *Information Systems Research* **28**(1), 64–83.
- Germonprez, M., Levy, M., Kendall, J. E. and Kendall, K. E. (2020), ‘Tapestries of innovation: Structures of contemporary open source project engagements’, *Journal of the Association for Information Systems* **21**(3), 5.
- Gilbert, R. J. and Katz, M. L. (2001), ‘An economist’s guide to US vs Microsoft’, *Journal of Economic perspectives* **15**(2), 25–44.
- Grant, R. M. (1991), ‘The resource-based theory of competitive advantage: implications for strategy formulation’, *California management review* **33**(3), 114–135.
- Guimarães, A. L., Korn, H. J., Shin, N. and Eisner, A. B. (2013), ‘The life cycle of open source software development communities’, *Journal of Electronic Commerce Research* **14**(2), 167.
- Healy, K. and Schussman, A. (2003), The ecology of open-source software development, Technical report, University of Arizona, USA.
- Hecker, F. (1999), ‘Setting up shop: The business of open-source software’, *IEEE software* **16**(1), 45–51.
- Herstatt, C. and Ehls, D. (2015), *Open source innovation: the phenomenon, participant’s behaviour, business implications*, Routledge.
- Horn, F. (2004), *L’économie des logiciels*, repères, la Découverte.
- Izquierdo, J. L. C. and Cabot, J. (2018), The role of foundations in open source projects, in ‘IEEE/ACM 40th International Conference on Software

- Engineering: Software Engineering in Society (ICSE-SEIS)', IEEE, pp. 3–12.
- Jansen, S. (2014), 'Measuring the health of open source software ecosystems: Beyond the scope of project health', *Information and Software Technology* **56**(11), 1508–1519.
- Joblin, M., Eckl, B., Bock, T., Schmid, A., Siegmund, J. and Apel, S. (2023), 'Hierarchical and hybrid organizational structures in open-source software projects: A longitudinal study', *Transactions on Software Engineering and Methodology* **32**(4), 1–29.
- Jullien, N. and Roudaut, K. (2012), 'Can open source projects succeed when the producers are not users? lessons from the data processing field', *Management international = International management = Gestión internacional* **16**, 113–127.
- Jullien, N., Stol, K.-J. and Herbsleb, J. D. (2019), A preliminary theory for open-source ecosystem microeconomics, in B. Fitzgerald, A. Mockus and M. Zhou, eds, 'Towards Engineering Free/Libre Open Source Software (FLOSS) Ecosystems for Impact and Sustainability', Springer, pp. 49–68.
- Jullien, N. and Zimmermann, J.-B. (2009), 'Firms' contribution to open-source software and the dominant user's skill', *European Management Review* **6**, 130 – 139.
- Jullien, N. and Zimmermann, J.-B. (2011), 'Floss firms, users and communities: a viable match?', *Journal of Innovation Economics* **1**(7), 31–53.
- Kang, Y. and Son, S. (2016), 'An analysis of open source gis software business models and case studies', *Spatial Information Research* **24**, 745–754.
- Kendall, K. E., Kendall, J. E., Germonprez, M. and Mathiassen, L. (2020), 'The third design space: A postcolonial perspective on corporate engagement with open source software communities', *Information Systems Journal* **30**(2), 369–402.
- Koch, S. (2008), 'Effort modelling and programmer participation in open source software projects', *Information Economics and Policy* **20**(4), 345–355.

- Kogut, B. M. and Metiu, A. (2001), ‘Open source software development and distributed innovation’, *Oxford Review of Economic Policy* **17**(2), 248–264.
- Krishnamurthy, S. (2002), ‘Cave or community? an empirical examination of 100 mature open source projects’, *First Monday* **7**(6).
URL: <https://firstmonday.org/ojs/index.php/fm/article/view/1477>
- Kuwata, Y. and Miura, H. (2015), ‘A study on growth model of oss projects to estimate the stage of lifecycle’, *Procedia Computer Science* **60**, 1004–1013.
- Lakhani, K. and von Hippel, E. (2003), ‘How open source software works: Free user to user assistance’, *Research Policy* **32**, 923–943.
URL: <http://opensource.mit.edu/papers/lakhanivonhippelusersupport.pdf>
- Le, D. M., Link, D., Shahbazian, A. and Medvidovic, N. (2018), An empirical study of architectural decay in open-source software, *in* ‘2018 IEEE International conference on software architecture (ICSA)’, IEEE, pp. 176–17609.
- Lee, S., Baek, H. and Jahng, J. (2017), ‘Governance strategies for open collaboration: Focusing on resource allocation in open source software development organizations’, *International Journal of Information Management* **37**(5), 431–437.
- Lee, S., Baek, H. and Oh, S. (2020), ‘The role of openness in open collaboration: A focus on open-source software development projects’, *ETRI Journal* **42**(2), 196–204.
- Lehman, M. M. (1980), ‘Programs, life cycles, and laws of software evolution’, *Proceedings of the IEEE* **68**(9), 1060–1076.
- Li, J., Conradi, R., Bunse, C., Torchiano, M., Slyngstad, O. P. N. and Morisio, M. (2009), ‘Development with off-the-shelf components: 10 facts’, *IEEE software* **26**(2), 80–87.
- Li, X., Moreschini, S., Zhang, Z. and Taibi, D. (2022), ‘Exploring factors and metrics to select open source software components for integration: An empirical study’, *Journal of Systems and Software* **188**, 111255.
- Li, X., Zhang, Y., Osborne, C., Zhou, M., Jin, Z. and Liu, H. (2024), ‘Systematic literature review of commercial participation in open source software’, *ACM Trans. Softw. Eng. Methodol.* .
URL: <https://doi.org/10.1145/3690632>

- Liao, Z., Zhao, B., Liu, S., Jin, H., He, D., Yang, L., Zhang, Y. and Wu, J. (2019), 'A prediction model of the project life-span in open source software ecosystem', *Mobile Networks and Applications* **24**, 1382–1391.
- Lin, Y.-K. and Maruping, L. M. (2022), 'Open source collaboration in digital entrepreneurship', *Organization Science* **33**(1), 212–230.
- Linåker, J. and Regnell, B. (2020), 'What to share, when, and where: balancing the objectives and complexities of open source software contributions', *Empirical Software Engineering* **25**, 3799–3840.
- Lindberg, A., Berente, N., Gaskin, J. and Lyytinen, K. (2016), 'Coordinating interdependencies in online communities: A study of an open source software project', *Information Systems Research* **27**(4), 751–772.
- Liu, M., Hansen, S. and Tu, Q. (2021), 'Sustaining collaborative software development through strategic consortium', *The Journal of Strategic Information Systems* **30**(3), 101671.
- Lundell, B., Gamalielsson, J., Tengblad, S., Yousefi, B. H., Fischer, T., Johansson, G., Rodung, B., Mattsson, A., Oppmark, J., Gustavsson, T. et al. (2017), Addressing lock-in, interoperability, and long-term maintenance challenges through open source: How can companies strategically use open source?, in 'Open Source Systems: Towards Robust Practices: 13th IFIP WG 2.13 International Conference, OSS 2017, Buenos Aires, Argentina, May 22-23, 2017, Proceedings 13', Springer International Publishing, pp. 80–88.
- MacCormack, A., Rusnak, J. and Baldwin, C. Y. (2006), 'Exploring the structure of complex software designs: An empirical study of open source and proprietary code', *Management Science* **52**(7), 1015–1030.
- March, J. G. (1991), 'Exploration and exploitation in organizational learning', *Organization science* **2**(1), 71–87.
- Marwell, G. and Oliver, P. (1993), *The Critical Mass in Collective Action: A Micro-Social Theory*, Cambridge University Press, Cambridge.
- Massa, L., Tucci, C. L. and Afuah, A. (2017), 'A critical assessment of business model research', *Academy of Management Annals* **11**(1), 73–104.

- McIntyre, D., Srinivasan, A., Afuah, A., Gawer, A. and Kretschmer, T. (2021), ‘Multisided platforms as new organizational forms’, *Academy of Management Perspectives* **35**(4), 566–583.
- Medappa, P. K. and Srivastava, S. C. (2019), ‘Does superposition influence the success of floss projects? an examination of open-source software development by organizations and individuals’, *Information Systems Research* **30**(3), 764–786.
- Mockus, A., Fielding, R. and Herbsleb, J. D. (2000), A case study of open source software development: The apache server, *in* ‘Proceedings’, The International Conference on Software Engineering (ICSE’2000), Limerick, Ireland, pp. 263–272.
- Moon, E. and Howison, J. (2024), ‘A dynamic perspective on software modularity in open source software (oss) development: A configurational approach’, *Information and Organization* **34**(1), 100499.
- Morgan, L., Feller, J. and Finnegan, P. (2013), ‘Exploring value networks: theorising the creation and capture of value with open source software’, *European journal of information systems* **22**, 569–588.
- Morgan, L. and Finnegan, P. (2014), ‘Beyond free software: An exploration of the business value of strategic open source’, *The Journal of Strategic Information Systems* **23**(3), 226–238.
- Mouakhar, K. and Benkeltoum, N. (2020), ‘Capacité d’absorption des entreprises de l’open source: du modèle d’affaires à l’intention d’affaires’, *Systèmes d’Information et Management* **25**(1), 47–88.
- Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K. and Ye, Y. (2002), Evolution patterns of open-source software systems and communities, *in* ‘Proceedings’, The International workshop on Principles of software evolution, pp. 76–85.
- Nguyen Duc, A., Cruzes, D. S., Hanssen, G. K., Snarby, T. and Abrahamsson, P. (2017), Coopetition of software firms in open source software ecosystems, *in* ‘Proceeding 8’, Software Business: 8th International Conference, Essen, Germany, pp. 146–160.

- Noureddine, A. (2023), ‘Red hat and its clones: Keep calm’.
URL: <https://www.noureddine.org/articles/red-hat-and-its-clones-keep-calm>
- Olson, D. L., Johansson, B. and De Carvalho, R. A. (2018), ‘Open source erp business model framework’, *Robotics and Computer-Integrated Manufacturing* **50**, 30–36.
- O’Mahony, S. (2003), ‘Guarding the commons: how community managed software projects protect their work’, *Research Policy* **32**(7), 1179 – 1198. Open Source Software Development.
- O’Mahony, S. and Ferraro, F. (2007), ‘The emergence of governance in an open source community’, *Academy of Management Journal* **50**, 1059–1106.
- Orucevic-Alagic, A. and Höst, M. (2014), Network analysis of a large scale open source project, in ‘40th EUROMICRO Conference on Software Engineering and Advanced Applications’, IEEE, pp. 25–29.
- Osterwalder, A. and Pigneur, Y. (2010), *Business model generation: a handbook for visionaries, game changers, and challengers*, John Wiley & Sons.
- Palazzi, M. J., Cabot, J., Canovas Izquierdo, J. L., Solé-Ribalta, A. and Borge-Holthoefer, J. (2019), ‘Online division of labour: emergent structures in open source software’, *Scientific reports* **9**(1), 13890.
- Pisano, G. P. and Teece, D. J. (2007), ‘How to capture value from innovation: Shaping intellectual property and industry architecture’, *California management review* **50**(1), 278–296.
- Raymond, E. S. (1999), *The Cathedral & the Bazaar; Musing on Linux and Open Source by Accidental Revolutionary*, O’Reilly, Sebastopol, Calif.
- Riehle, D. (2012), ‘The single-vendor commercial open course business model’, *Information Systems and e-Business Management* **10**, 5–17.
- Riehle, D. (2021), ‘The open source distributor business model’, *Computer* **54**(12), 99–103.
- Rogers, E. M. (2010), *Diffusion of innovations*, Simon and Schuster.

- Rosenberg, N. (1982), *Inside the black box: technology and economics*, Cambridge University Press.
- Ross, J. W., Beath, C. M. and Goodhue, D. L. (1996), ‘Develop long-term competitiveness through it assets’, *Sloan management review* **38**(1), 31–42.
- Sánchez, V. R., Ayuso, P. N., Galindo, J. A. and Benavides, D. (2020), ‘Open source adoption factors—a systematic literature review’, *IEEE Access* **8**, 94594–94609.
- Schaarschmidt, M. (2023), ‘Innovating beyond firm boundaries: resource deployment control in open source software development’, *Information Technology & People* **36**(4), 1645–1668.
- Schaarschmidt, M., Walsh, G. and von Kortzfleisch, H. F. (2015), ‘How do firms influence open source software communities? a framework and empirical analysis of different governance modes’, *Information and Organization* **25**(2), 99–114.
- Setia, P., Bayus, B. L. and Rajagopalan, B. (2020), ‘The takeoff of open source software: A signaling perspective based on community activities’, *MIS quarterly* **44**(3).
- Shahrivar, S., Elahi, S., Hassanzadeh, A. and Montazer, G. (2018), ‘A business model for commercial open source software: A systematic literature review’, *Information and Software Technology* **103**, 202–214.
- Shaikh, M. and Cornford, T. (2011), Total cost of ownership of open source software: a report for the UK Cabinet Office supported by OpenForum europe, Technical report, UK Cabinet Office.
- Shaikh, M. and Henfridsson, O. (2017), ‘Governing open source software through coordination processes’, *Information and Organization* **27**(2), 116–135.
- Shaikh, M. and Levina, N. (2019), ‘Selecting an open innovation community as an alliance partner: Looking for healthy communities and ecosystems’, *Research Policy* **48**(8), 103766.
- Shaikh, M. and Vaast, E. (2023), ‘Algorithmic interactions in open source work’, *Information Systems Research* **34**(2), 744–765.

- Simões, C. A. and Santos, G. (2024), It workforce outsourcing benefits, challenges and success factors-systematic mapping study, *in* ‘Proceedings of the 20th Brazilian Symposium on Information Systems’, ACM, pp. 1–10.
- Spijkerman, Z. and Jansen, S. (2018), The open source software business model blueprint: A comparative analysis of 10 open source companies., *in* ‘SiBW’, pp. 128–143.
- Stamelos, I., Varlamis, I., Anagnostopoulos, D. and Gonzalez-Barahona, J. M. (2020), ‘Open source systems: Enterprise software and solutions’, *Journal of Systems and Software* **163**, 110541.
URL: <https://www.sciencedirect.com/science/article/pii/S0164121220300236>
- Stol, K.-J., Avgeriou, P., Babar, M. A., Lucas, Y. and Fitzgerald, B. (2014), ‘Key factors for adopting inner source’, *ACM Transactions on Software Engineering and Methodology (TOSEM)* **23**(2), 18.
- Stol, K.-J., Babar, M. A., Avgeriou, P. and Fitzgerald, B. (2011), ‘A comparative study of challenges in integrating open source software and inner source software’, *Information and Software Technology* **53**(12), 1319–1336.
- Sutanto, J., Jiang, Q. and Tan, C.-H. (2021), ‘The contingent role of inter-project connectedness in cultivating open source software projects’, *The Journal of Strategic Information Systems* **30**(1), 101598.
- Tadelis, S. (2007), ‘The innovative organization: Creating value through outsourcing’, *California Management Review* **50**(1), 261–277.
- Teece, D. J. (2010), ‘Business models, business strategy and innovation’, *Long range planning* **43**(2-3), 172–194.
- Teece, D. J. (2018), ‘Profiting from innovation in the digital economy: Enabling technologies, standards, and licensing models in the wireless world’, *Research policy* **47**(8), 1367–1387.
- Teixeira, J., Mian, S. and Hytti, U. (2016), Cooperation among competitors in the open-source arena: The case of openstack, *in* ‘International Conference on Information Systems’, Dublin, Ireland.
- Thiruthy, N. (2017), ‘Open source—is it an alternative to intellectual property?’, *The Journal of World Intellectual Property* **20**(1-2), 68–86.

- Tilson, D., Lyytinen, K. and Sørensen, C. (2010), ‘Research commentary – digital infrastructures: The missing is research agenda’, *Information systems research* **21**(4), 748–759.
- Trzeciak, M., Sienkiewicz, Ł. D. and Bukłaha, E. (2022), ‘Enablers of open innovation in software development micro-organization’, *Journal of Open Innovation: Technology, Market, and Complexity* **8**(4), 174.
- Tuomi, I. (2001), ‘Internet, innovation, and open source: Actors in the network’, *First Monday* **6**(1).
URL: http://www.firstmonday.org/issues/issue6_1/tuomi/index.html
- Van der Linden, F., Lundell, B. and Marttiin, P. (2009), ‘Commodification of industrial software: A case for open source’, *IEEE software* **26**(4), 77–83.
- Viseur, R. and Charleux, A. (2021), Open Source Communities and Forks: A Rereading in the Light of Albert Hirschman’s Writings, in ‘17th IFIP International Conference on Open Source Systems (OSS)’, pp. 59–67.
- Viseur, R. and Jullien, N. (2023), ‘Communesplone, an original open-source model of resource pooling in the public sector.’, *IEEE Software* **40**, 46–54.
- von Hippel, E. (2001), ‘Innovation by user communities: Learning from open-source software’, *MIT Sloan management review* **42**(4), 82–86.
- von Hippel, E. and von Krogh, G. (2003), ‘Open source software and the "private-collective" innovation model: Issues for organization science’, *Organization Science* **14**(2), 209–223.
- Wankhede, P., Talati, M. and Chinchamalature, R. (2020), ‘Comparative study of cloud platforms-Microsoft Azure, Google cloud platform and Amazon ec2’, *J. Res. Eng. Appl. Sci* **5**(02), 60–64.
- Weeks, M. R. and Feeny, D. (2008), ‘Outsourcing: From cost management to innovation and business value’, *California management review* **50**(4), 127–146.
- Weng, Q. and Soh, F. (2023), ‘The influence of project initiators’ person-to-person followership on project popularity in open source communities: The role of reach and importance’, *The Journal of Strategic Information Systems* **32**(2), 101771.

- West, J. (2003), ‘How open is open enough? Melding proprietary and open source platform strategies’, *Research Policy* **32** (7), 1259–1285.
- West, J. and Dedrick, J. (2001), ‘Open source standardization: The rise of linux in the network era’, *Knowledge, Technology & Policy* **14**(2), 88–112.
- West, J. and Gallagher, S. (2006), ‘Challenges of open innovation: the paradox of firm investment in open-source software’, *R&D Management* **36**(3), 319–331.
- Yost, J. R. (2017), *Making IT Work: A History of the Computer Services Industry*, MIT Press.
- Zhang, Y., Zhou, M., Mockus, A. and Jin, Z. (2021), ‘Companies’ participation in oss development—an empirical study of openstack’, *IEEE Transactions on Software Engineering* **47**(10), 2242–2259.
- Zimmermann, T. (2020), A first look at an emerging model of community organizations for the long-term maintenance of ecosystems’ packages, *in* ‘Proceedings of the 42nd International Conference on Software Engineering Workshops’, IEEE/ACM, pp. 711–718.