Linnéuniversitetet
Kalmar Växjö

Master Thesis Project

Software Architecture for a Cyber-Physical Ecosystem in support of Open Innovation

- Balancing Open Innovation and Governance through Software Architecture

Author: Efthymios Platanias
Supervisor: Jesper Andersson
Examiner: Welf Löwe
Reader: Jonas Lundberg
Semester: VT 2017
Course Code: 4DV50E
Subject: Computer Science

Abstract

This is a qualitative exploratory study of Software Architecture in Cyber-Physical Ecosystems. Software Architecture plays a pivotal role to Software Ecosystems and, apart from Functional and non-Functional requirements, it is affected by factors of a different nature. The purpose of this paper is to identify these factors and describe their relationship with the ecosystem's architecture. Several owners of Cyber-Physical systems are in the process of setting up new ecosystems by sharing functionalities of their proprietary platform with third-party developers. This makes Architecture that supports Open Innovation critical to this endeavor. We believe that the application of Software Ecosystem best practices to the domain of Cyber-Physical Systems is an interesting subject. An exploratory literature study was conducted to create a conceptual model which describes the relationship of architecture with the factors presented above. This study resulted in a conceptual model which supports the decision making process of the platform owner during the various stages of the ecosystem's lifecycle.

Keywords: Software Architecture, Software Ecosystems, Cyber-Physical Systems, Open Innovation

Preface

I would like to thank the course administrator Dr Narges Khakpour and my supervisor Dr Jesper Andersson for their support and guidance, and my partner Dr Ekaterini Drosou for her patience with me.

Contents

List of Figures

List of Tables

1. Introduction

Software Ecosystems are a natural evolution of Software Product Lines and are becoming increasingly popular as a business model. Apple's iOS and Google's Android are two of the most well-known examples [1]. Software Architecture is one of the most critical aspects concerning Software Ecosystems. In this paper, we are interested in studying how does Software Architecture affects the ecosystem and vice-versa.

The phenomenon of ubiquitous computing is becoming a reality in the modern world at an impressively fast rate. Because of this, software is becoming a key-enabler and value creator in non-traditional markets and companies which traditionally were not in the Software Business, suddenly realize that this is not the case any further, something which might require a slight change of mindset. One prominent example of this situation is owners of Cyber-Physical Systems.

The increasing popularity of Cyber-Physical Systems provides opportunities in the form of business ecosystems, and companies are now beginning to realize the potential of such set-ups. In this project, we are interested in studying how ecosystem platforms for Cyber-Physical Ecosystems should be designed. Specifically, we are interested in identifying and categorizing the decisions software architects must take in the design phase of such a platform, decisions that will be imprinted in the whole life-cycle of the platform created.

A qualitative description of what choices should a Software architect take during the design of a platform and the tradeoffs that are involved in this process, will be given. In this way, we aim to aid in enriching the knowledge base of Abstractions and Interfaces in the field of Platform Ecosystems in the domain of Cyber-Physical Systems, which in turn will further provide help to boost innovation in the field [2]. Progress in this field might prove useful to the industry, as Cyber-Physical Systems have an increasing number of applications in the field.

A Software Ecosystem based on a platform in this setting would enable open innovation by third party developers, who in turn can provide highly specialized applications to satisfy different customer segments, while sharing risks with the company that owns the platform (further on referred to as the platform owner) for little or no cost.

However, creating a platform design is not a trivial task because, even though it seems like a technical decision, it is also influenced by other, business strategy related, factors. One of the most prominent factors, that a software architect should be thoughtful of when designing a platform, is the rapid evolution of technology and innovations that are expected to change the field of computing radically, with Cyber Physical Systems (CPS) and the Internet of Things (IoT) being an example. A platform's ability to evolve, strongly influences the survivability of the Ecosystem built around it.

1.1. Motivation

We believe that this study is important for the following reasons:

- Several companies are transitioning from traditional manufacturing of physical equipment to providers of Cyber-Physical Systems. This situation creates new opportunities for companies to set up new ecosystems based on the data that could be exploited in CPS through products or services. However, the area of Software Ecosystems is still in its infancy even though it is becoming increasingly popular since the first publication relevant to it at 2005. Even though there has been an increased number of research from 2007 and the domain of Software Ecosystems has now a dedicated workshop (IWSECO) [32], there is still need for further research in order to better understand how do architectures that foster open- innovation, which is a critical feature of any viable software ecosystem nowadays, look like [4].
- Furthermore, there is a relatively small number of research that relate to the Industry and take place in a real-world scenario, and most of them are also concerned with Free or Open Source Software (FOSS) Ecosystems, with the other choice being proprietary software Ecosystems. While studies relative to FOSS ecosystems are mostly concerned with technological or social matters, studies concerned with proprietary ecosystems mostly include business and strategic problems [32]. In the previous sections we discussed about how software architecture plays a critical strategic role in platform ecosystems, and why keystone firms (platform owners) should pay attention to decisions relevant to it. The number of experience studies in the field of software ecosystems that also provide a real-world solution is really small. Only 2% of studies mapped in [4] and 4% of studies reviewed in [32] include a solution to a specific problem.
- Given the rising popularity of software ecosystems and the crucial role of software architecture, we believe that it is highly important to understand how to design and build platforms that enable open innovation while achieving a fine balance in several qualities that include trade-offs like performance, maintainability, resilience, evolvability, modularity etc. We also believe that phenomena new to the world of computing like Cyber Physical Systems and their Data-Intensive nature, are bound to affect our design decisions providing interesting insights for ecosystems that are built around CPS platforms.
- Some studies [11] [26] [27]have addressed Cyber Physical Systems in manufacturing and industry, but without mentioning the possibility of setting up an ecosystem, based on a platform which is part of their respective CPS. Software Ecosystems, especially those built around a proprietary platform, are in the intersection of Business, Open Innovation and Software Engineering. Thus, their requirements are also dictated by the strategic motives of the firm. Even though multi-sided platforms have been discussed in many domains, there is an absence of attention to this matter in the domain of CPS.

For the reasons presented above, we will conduct a study in a real-world scenario in an industrial setting for platforms of a proprietary nature. In this study, we are interested in proprietary software ecosystems and we will use a conceptual

model to provide support and describe the multifaceted nature of decisions need to be taken to design platform that is supposed to enable open innovation in a flexible but also stable manner. We adopt the following definition for flexibility of the platform. “Flexibility: The ease with which a system or component can be modified for use in applications or environments other than those for which is was specifically designed.” [33]. For this, we will conduct a rigorous literature study on sources relevant to real world software ecosystems in the domain. Our main interest, is to learn more about how should a platform be designed in order to provide enough autonomy to the app developers without compromising integration. By cross-checking our model with the concerns of the companies, we aim to learn from this experience, about all choices that will have to be made, concerning modularization, interface standardization and other still unknown factors that are bound to influence our decisions.

1.2. An Illustrative Example

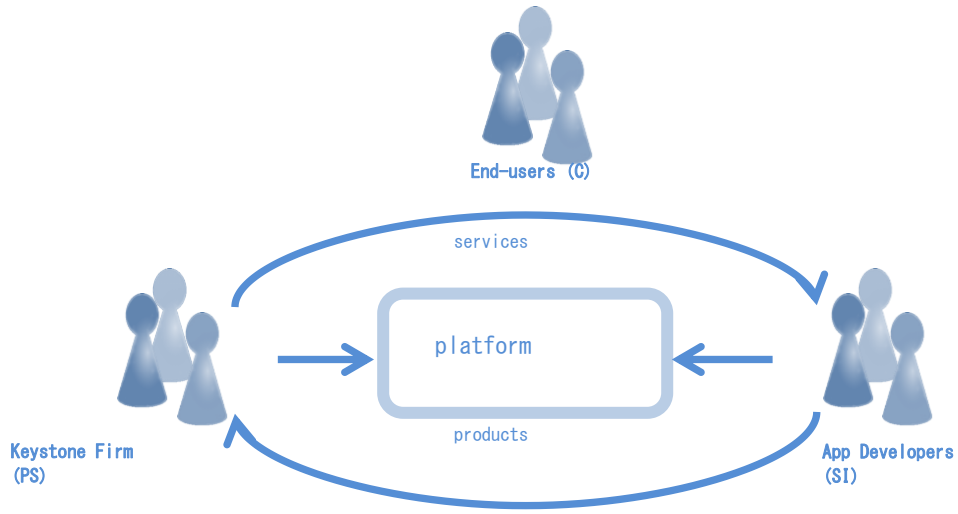

Figure 1 The PS Ecosystem

To guide the reader in understanding the concepts related to the problem we will use an illustrative example throughout the whole paper to explain concepts, decisions and strategies. Company PS is a platform owner in a CPS. Its main business is supplying robotic parts to customers (C) which are other Industrial companies (e.g. owning manufacturing lines). The role of the app developers is held by the System Integrators (SI) whose main business is developing complementary software for the parts supplied by the PS, creating systems of these modules with unique emergent properties, and selling them to the customers. The keystone firm (PS) and its several System Integrators (SI) form a Software Ecosystem. The survivability of this ecosystem is dependent on decisions made by PS, concerning several aspects. These aspects serve the Ecosystem’s strategic goals which are driven by its Lifecycle. These concepts will be explained in detail in later sections.

A critical mission of PS is to amass adopters of its proprietary platform in both customer segments (C and SI). This will create strong network effects, and the

value of the platform will rise exponentially [1]. Furthermore, strong network effects are crucial for several evolutionary attributes of the platform, as will be explained later.

1.3. Problem Statement and Research Questions

The object of inquiry in this project is the Software Architecture of a platform for a Cyber-Physical Ecosystem. This leads to a straightforward research question. The research question is:

How should a platform offering Data-Intensive Services in a Cyber Physical Ecosystem be designed in order to enable open innovation without compromising integration?

In order to answer, though, this question we must first discover the questions it is comprised of. To create a new software design that would come up as a novel contribution to a domain, a software engineer must first know its requirements and also have an idea of what is needed [3].

Furthermore, a factor that plays a key role in the design we are interested in is a platform's modularity. Even though modularity seems like a prequisite when trying to achieve high evolvability/flexibility with minimal integration costs it involves trade-offs as it initially performs worse than monolithic architectures and at a higher cost.

Thus, we will investigate two more specific questions:
What should this type of software platform support? Specifically, what requirements enable Open Innovation with minimal Integration Costs?
How is the platform designed? What 's its architecture?

This leads to the following sub-questions, which, when answered, provide the answer to my main research question described above.

RQ1	What are the requirements of an ecosystem platform for Data-Intensive Services in a Cyber-Physical Ecosystem which supports open innovation?
RQ2	What architectural decisions would enable these requirements?

Table 1: Research Questions

1.4. Contributions and Limitations

a. Contributions

Answering this question provided a qualitative description of how Software Architecture and design affects and is affected by several factors in addition to the Functional and Non-Functional Requirements of a Platform. Furthermore, we studied the disperse domains which are involved in Software Ecosystems with the intention to identify best practices from the Real World. Our result was a conceptual model that in addition of answering our research questions also supports the

platform owner to take decisions concerning the design of the ecosystem. The Decision Support Model by itself is also a contribution, as it is a usable conceptual artifact which can support the decision-making process of the platform owner.

b. Limitations and Challenges

The limitations of this study are directly and indirectly related to the challenges that we faced under the duration of this study.

The first challenge was the multiplicity of scientific domains that the subject of this study involves. The writer had to orient himself in five different scientific domains as Cyber-Physical Ecosystems lie on the intersection of all research domains presented above. This challenge, combined with the tight timeframe, presents the limitation of not having a complete coverage of literature for all domains.

The second challenge was the remoteness of some scientific domains with the domain of Computer Science, which is our background. We approached concepts like Business Models, Innovation and Business management from a Computer Science perspective. Thus, the author's background might affect the way these concepts were interpreted during the study.

Another challenge and limitation of this study is that, to the best of our knowledge, there is no literature covering Cyber Physical Ecosystems. Even though Cyber Physical Systems was the key motivator of this study, the absence of sources resulted in a decision support model which covers the broader research domain of Software Ecosystems. This is reflected on the result of this study, the Decision Support Model. However, based on the Data Intensive nature of this domain, it is rather safe to assume that scalability is of vital importance to Cyber Physical Ecosystems. Thus, the way the platform is modularized is connected, in this way, to the Cyber Physical Domain. The author believes firmly that this model applies also to Cyber Physical Ecosystems because of their inherent emphasis on Software, hence the "cyber" term.

Another limitation of this study is the absence of evaluation. In order to evaluate the conceptual artifact that is the result of this study, longitudinal studies would be the best option, something which does not fit the given timeframe. Then, we would have to choose between two options: First, to use qualitative methods (interviews) to evaluate the model, or to finalize and improve the model itself. We chose the latter because interviews would have been weak concerning validity and the model's quality would suffer as efforts should be also focused on collecting and analyzing data. Thus, we focused our efforts in producing a decision support model of better quality and we leave the evaluation as reference for future work and development of it.

1.5. Scientific Approach

This is a qualitative exploratory literature study with a deducting approach. The philosophical stance adopted for this study is Pragmatism, as we are interested in what works in real-life. We aim to create a classification of the requirements that a platform owner must fulfill in order to create and manage a sustainable platform.

1.6. Method Description

The method used in this thesis is Literature Study. This method is intended to provide answers to the research question. It is an exploratory qualitative study, as our intention is to seek new insights about the requirements of the platforms of interest [48].

- Goal: To identify state-of-the-art and state-of-practice concerning Platform Design for platforms that support Open Innovation.
- Question: What are the special requirements for Software Ecosystems which support Open Innovation?
- Result: A conceptual model which summarizes an supports the decisions that a platform owner needs to take in the various stages of an ecosystem's Lifecycle, concerning various views.

Using pragmatism as theoretical lenses in our literature study, we filter information and actively look for best practices by real-world companies. Studies including real cases are in the focus of the literature study in addition to books and papers inspired (and sometimes presenting) real world cases.

Existing systems, platforms, tactics and strategies are studied concerning Software Ecosystems which enable and support Open Innovation. This contributes in a better understanding of the design concerns that are relevant to platforms which support Open Innovation.

1.7. Results

The result of our study is a conceptual model which in addition to describing the various special requirements also supports the decision maker in all relevant fields, platform architecture included. This model describes how the business nature of software ecosystems affects technical decisions, and how all the dimensions of a platform ecosystem described in Chapter 2 should be coherent with each other for a successful platform ecosystem.

1.8. Target group

This study is firstly interesting for companies which are the owners of platforms open to third-party developers, in any context. The decision support model by itself provides useful information to the decision maker within the keystone firm. It also has interest for individuals within the domains of Software Architecture and Engineering like Software Architects and System Engineers, as it describes the relationship of Architecture to factors additional to Functional and Non-Functional requirements, which have a big impact on it. Furthermore, this study is also interesting to researchers within the domain of Knowledge Management and especially Open Innovation, because Software Ecosystems are the broadest form of Open Innovation which is concerned about knowledge inflows and outflows of the keystone firm.

1.9. Report Structure

The rest of the document is structured as follows: Chapter 2, gives a detailed description of the scientific background for our research and introduces the reader to the several factors that affect the architecture of the platform, and the best practices concerning those, according to the literature. The state of the art concerning the field of Software Ecosystems is also presented to the reader. The Background section is structured also in a way that can be easily referenced upon, because we discovered that it was needed in order to make our Decision Support Model easier to use and more understandable. In chapter 3, we present the result of our study, the Decision Support Model. The fourth chapter of this paper presents the methods we used to derive the information needed to compile our model and the last chapter concludes this paper and gives suggestions for future work.

2. Background

This project is relevant with several domains in Computer Science, including Software Architecture, Software Ecosystems and Cyber Physical Systems. Each one of them will be presented in this section.

2.1. Software Ecosystems

A Software Ecosystem is created when a company that owns a software platform (the architecture of the product line and the shared components), decides to open it up to third party developers who will create new customized versions of it by developing complementary functionality (from here on referred to as apps) [3].

Specifically, a company transitions from a software product line to a software ecosystem as soon as the scope of the product line exceeds the company's boundaries. Bosch [3] identifies two reasons for this action. First, to satisfy functionality requirements that would not be possible in reasonable time and R&D resources, and build a wider customer base than the one it would build in the same amount of time without the help of third party developers. Secondly, transitioning to a software ecosystem is a way for a company to achieve massive customization of its product line.

A platform-based Software Ecosystem can be divided into two broad subsystems, the platform and the complementary applications (apps). The firm which owns the platform is called the platform owner or keystone firm. It is the firm that wants to expand, adding new functionality to its own software (the platform). Instead of developing every application inside the firm, ecosystems offer the opportunity to the firm to harness the massive innovative potential of third party developers, each one with individual interests and another point of view to the market [1]. The platform then, is the central part of an Ecosystem which provides the foundations to third party developers to create specialized solutions as they see fit.

One key challenge concerning the platform owner is how to create a platform that offers enough freedom to the app developers to use their creativity and experiment to come up with interesting solutions, while achieving seamless integration between the platform and apps. In order to keep the whole endeavor organized and avoid big coordination costs, the notion of platform governance enters the picture. Governance, in short terms, defines who decides what in a platform ecosystem [1]. Governance is concerned with how decision rights are divided between platform owner and the app developer, the control mechanisms, formal or informal, that are used by the platform owner for coordination (control portfolio) and pricing structures [1].

A platform owner must cooperate with an increasingly high number of individual app developers, something which makes coordination costs very high and traditional means of control and governance useless or harmful to the evolutionary potential of the ecosystem [4]. Making contribution from third party developers as simple as possible through "the use of generic and popular environments and stable and expressive interfaces" is one of the most important challenges a keystone firm will face [3]. The same points are also brought up in [1], especially concerning making the life of app developers easier and reducing

complexity by restricting all communication between the platform and the app developers to the specified interfaces, which are supposed to also be stable and precisely documented [1].

Thus, the platform owner must find the right balance of control and freedom provided to the app developers, so that the latter can innovate undistracted without compromising cohesiveness along the platform. The solution to this emerging problem is for the platform owner not to try and control the app developers, but orchestrate them [1]. Software Architecture has a pivotal role concerning the orchestration of Innovation from the app developers. Thus, Software Architecture becomes a critical aspect of Software Ecosystems, and acts as a foundation over which Ecosystem Governance is built. It is presented in the next section.

2.2. Software Architecture

Software Architecture is an abstraction of a system describing how it is structured, its components, their properties and their behavior. It encompasses all principal design decisions that are made when a system is created [5]. It defines how it is created and how it will evolve, and it also acts as a blueprint for multiple related systems, like a product family or product line, through the notion of reference architecture. These design decisions involve the system structure, its functional behavior, the way it interacts and also its non-functional properties (like scalability and dependability) and its implementation [5].

Software Architecture may be developed with a focus on quality that is to satisfy certain quality attributes. Architecture tactics are architecture building blocks developed from experience to satisfy a specific quality attribute [6]. Several quality attributes have been identified like availability, performance, security and modifiability.

The importance of Software Architecture in business regarding product lines and productivity increases as it simplifies development, reduces development time and cost and also increases reliability through code reuse. Moreover, "Software Architecture provides the critical abstractions that enable variation and commonality with a product family to be simultaneously managed" [5].

Software Architecture seems to play a very important role to platform-based Software Ecosystems in general, as it includes decisions that affect several inherent properties of the platform, with the most important, when it comes to Ecosystems, affecting its complexity, how easy is it for the app developers to enhance the functionality of the platform by developing apps and in extent on how easy can this platform evolve [1].

Architecture defines how innovation work is divided between the platform owner and the app developer, and also how integration between their work outputs is achieved. Architecture, thus, reduces complexity for every participant of the ecosystem as it allows each side to focus on its job unconcerned about the other participants [1].

A system's complexity is analogous to the number of its subsystems and is also affected by the fact that a system's complexity grows over time. Complexity is the biggest enemy of an ecosystem as it can increase its incomprehensibility and create a gridlock [1]. A gridlock, is a situation where a change to a small part of the ecosystem, e.g an app, would have so serious consequences to the whole ecosystem (through a high number of dependencies) that it could break it. Furthermore, an

incomprehensible platform requires more effort for a developer to understand it and thus provide useful innovational solutions, whether by addition of new functionality (app) or its modification/evolution (platform developer).

The only solution against complexity is to reduce it [1]. Platform ecosystems have two types of complexity, structural and behavioral, with the first concerned on how difficult it is to describe the interconnections between its components, and the later concerning how difficult it is to predict or control its aggregate behavior [1]. Through architecture we can reduce structural complexity and through governance we can reduce the behavioral complexity [1].

Architecture can be a tool to describe the ecosystem's components and their relationship reducing its structural complexity, thus making it more manageable. Reducing the interdependencies between the different subsystems (apps and platform) when designing a platform ecosystem can make coordination easier between the different stakeholders of the ecosystem, while also boosting the effectiveness of each individual app, as the app developer can have the valuable ignorance that allows him to focus on his solution.

A key theme concerning Architecture and something that will be of interest for us in this paper is modularity. Modularity is a property of any complex system [1] and it is relevant to how divisible is a system to subsystems and also how much interdependent are these subsystems. Modularization of a system is the action of intentionally reducing the dependencies between its subsystems. Modularity, concerning a software-based platform ecosystem, is the degree on which its subsystems are independent of each other. This means that the platform and the apps can be designed, implemented and operated with minimal to no constraints from each other [1]. In a highly modular software ecosystem, the platform and the apps can be designed and implemented in any fashion that their developers see fit with only one constraint: to comply with a set of standardized interfaces through which they interact with the other components of the ecosystem.

However, there are also other factors affecting Software Architecture in the scope of this thesis. Cyber Physical Systems, which provide the context for this thesis, and Data-Intensive Applications, which are naturally a part of CPS, affect both Software Architecture and Governance.

2.3. Data-Intensive Applications

Data-Intensive Software Applications are software applications which involve collection, processing and dissemination of massive volumes of data. Data-Intensive Software is becoming increasing popular [7] and, due to the uniqueness of its nature, it creates new challenges in the Domain of Software Architecture, as existing engineering techniques and architecture techniques must be updated to accommodate the requirements that might emerge [7].

Cyber-Physical Systems fall into the category of Data-Intensive Software Systems as they have the capability of providing a massive stream of information through feedback loops concerning sensors or other instruments.

Data-Intensive applications in their architectural aspect have been receiving attention for more than a decade. Most of the research found that is related to Data-Intensive applications and ecosystems is in the domain of Science applications, as it traditionally has to deal with voluminous amounts of Data coming from complex experiments conducted by geographically dispersed groups of scientists [8].

Examples of such studies which resulted in an architectural style are [9] and [10]. Studying these papers provides an understanding on quality concerns and requirements that are inherent to Data-Intensive Applications in other domains, like the one we are interested in this study, CPE.

Even though the Data-Intensive nature of Cyber Physical Systems is not the main focus of this research, it is expected to affect several decisions regarding the actual architecture of the platform. For this reason, it is highly likely that cloud technologies could be a part of the design. Cloud-based architectures and frameworks have been recently emerging, also in the domain of Cyber Physical Systems due to the increasing importance of Data-Intensive Applications in it [11] [12]. One telling example of such research is [11], which is also in the Industrial domain which is of interest in this study. However, [11], focuses the integration of CPS with cloud-based technologies, and does not address the unique challenges that Software platform-based ecosystems entail. As explained before, design decisions in proprietary platforms are heavily affected by strategic business factors, as some form of governance must be engraved into the architecture of the system.

2.4. Cyber-Physical Systems

A domain relevant to Data-Intensive Software Systems and the one that will provide the context of this project is the domain of Cyber-Physical Systems. The term Cyber-Physical Systems is used to describe a new generation of systems that have integrated computational and physical capabilities [2] and can interact by various ways with humans. The data-intensive nature of this field, lies in the fact that this type of systems may produce a vast amount of data through feedback loops through e.g. sensors.

Several opportunities and challenges exist in the field of CPS while research in this field is still in an early stage [2]. One of this challenges lies in the domain of Software Architecture as there is an urgent need for standardized abstractions and architectures to support integration and interoperability and provide further support to innovation [2]. The increasing interest in Cyber-Physical Systems is also expressed by initiatives taken by E.U. the U.S. and other countries to support research and innovation on CPS [2]. Advances in research of CPS is bound to also benefit the industry as several grand challenges have been articulated in several industry sectors including Biomedical and Healthcare Systems [17], Advanced Air-Transportation and SmartGrid and renewable Energy [2].

While reviewing literature, the reader quickly understands that most of research and applications concerning CPS are in the domain of energy. Studies relevant to the SmartGrid and Electric Energy in general are the amongst the most popular results during a casual search of scientific work. Some examples are [18] [19] [20] [21]and [22]. Another popular subject in the domain of CPS is Security of CPS [23] [20]. Concerning System Architecture several examples of prototype architectures exist in the field [24] [25] with some of them relevant to manufacturing plants. Terms like “Factories of the Future” or the Industry 4.0 are quite popular and some researchers have addressed the application of CPS in the Industry [26] [27] [24]. A lot of these researches also include Cloud Computing and Data-Intensive Services [26] [17], something which shows the increasing importance of Data-Intensive Technologies for the domain of CPS.

However, no research was found that would present an architecture designed for a Software Ecosystem, one which balances the requirements for Open Innovation and Governance.

2.5. Open Innovation

Open Innovation is a paradigm relevant to how a firm should use external inflows of knowledge to boost its inner innovation processes. It is based on the assumption that firms 'can and should use external ideas as well as internal ideas, and internal and external paths to market, as they look to advance their technology' [28]. As a term it was first introduced by Chesbrough at 2003, who emphasized several factors which challenged the boundaries within which innovation takes place (traditionally innovation was residing inside the organizational boundaries, usually a R&D department) and boosted a move towards more open systems of innovation [29]. Since then, OI has been increasingly getting more attention by the Academia [29].

OI is a crucial concept when it comes to platform ecosystems because the main motivation for a company to make the transition to a platform ecosystem is to enhance the functionality of its own software product by incorporating ideas that originate from outside its organizational boundaries. Furthermore, Open Innovation is considered by other authors as one type of Inbound Innovation Strategies [30]. In [30], the authors introduce a typology of OI strategies, classified by the "breadth" and "depth" of innovation. Breadth refers to the diversity of external partners to the company, and depth to the level of integration of external knowledge given by selected partners to the company. OI is the most "open" type of Inbound OI strategies, as it involves a big number of diverse external partners (high breadth) who are deeply integrated with the company (high depth) [30]. In this study, we think of open innovation as the use of external and internal to the keystone firm knowledge, in order to enhance the functionality of the platform. We partly agree with the position presented in [30], which supports that an Ecosystem is an OI Strategy. However, we believe that Open Innovation is a part of an ecosystem's Strategy because OI is concerned with the exchange of knowledge and co-creation of value across the ecosystem, but an ecosystem's strategy is also concerned with value capture, something which is named as "Open Strategy" in [31]. Simply put, open innovation is concerned with how the ideas of third-party developers are adopted and integrated into the proprietary software product of the keystone firm (the platform).

Thus, the domains of Open Innovation and Software Ecosystems, CPS and Data-Intensive Applications, affect the decisions of the architect responsible for the design of the platform. It is in our interest to study how exactly the Software Architecture affects and is affected by these domains.

3. Decision Making in Software Ecosystem Design

Designing a platform for a Software Ecosystem entails decisions which are influenced by the many faceted nature of the domain [34]. There are several views of a platform ecosystem and each one of them affects and is affected by the others. We have categorized these views in the following groups: Open Innovation Strategy, Organizational Structure, Governance, Software Ecosystem Lifecycle and Software Architecture.

Below, we will attempt to present in detail each one of these different views that must be combined when making decisions concerning the platform's architecture, how each one of them interacts with the others and how does it affect the architecture. To promote clarity, we will use a top-down approach in our presentation starting from the SEco Lifecycle dimension and ending in the Software Architecture dimension.

The model presented in picture below, represents how these dimensions relate and interact with each other.

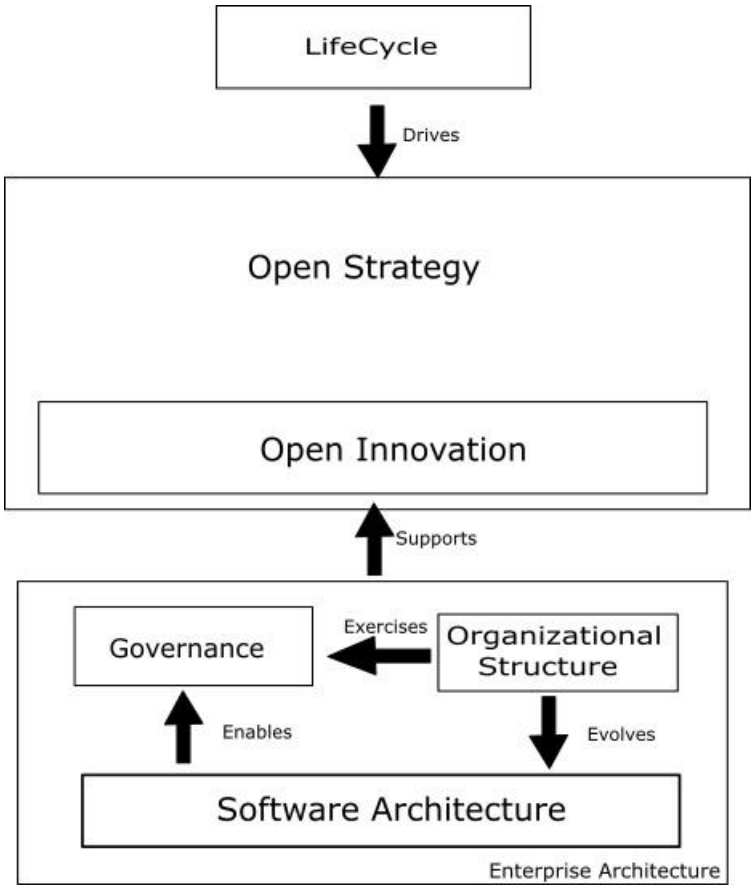

Figure 2: The five dimensions of a Software Ecosystem

The Case of PS Company

Based on the above, the strategy that PS follows is driven by its Lifecycle stage. This means that PS has different strategic goals and priorities in the beginning of its Lifecycle than the end. Thus, the Strategy of PS is dependent on the Lifecycle phase it is in. Furthermore, to achieve its strategic goals, PS must take specific decisions concerning the Architecture of its proprietary platform which will in turn support its Governance policies, used to guide its app developers (SI). Furthermore, to take decision regarding the Architecture and execute these Governance policies, PS must go through Organizational Restructuring and create Organizational Capabilities to fully take advantage of its architecture, harness the massive innovative potential of the SI, encourage more SI to adopt its platform and in turn get a bigger customer share (C) and gain a competitive advantage against rival platforms (Part Suppliers also).

3.1 Life Cycle

An ecosystem's Lifecycle and the stimuli from its external environment guide its strategy, like any organization [35]. Depending on the Lifecycle stage of the platform, the keystone firm should focus on different evolutionary properties [1] which, in turn, affects Governance, Structure and Architecture [36] [1].

3.1.1 The three Lifecycle dimensions

In this study, we adopt the three-dimensional approach which is used in [1]. We also believe that these three dimensions are interlinked to each other and happen in the same timeframe. The reader can refer to our model overview (*Figure 3: DSM and Lifecycle*) for each one the dimensions mentioned in the next sections.

3.1.1.1 The S-Curve

In [1], the author adopts a three-dimension view of the platform's Lifecycle. The first dimension is the maturity stage, or the phase of the platform along the S-Curve. This dimension is concerned with the market share of the platform, and how it evolves over time. While the stages are named differently across papers [36] [1], the four distinct phases in the S-curve are essentially the same. In this paper, we will adopt the approach taken in [1] and use the following names: Introduction, Ascent, Maturity and Decline. The phase of the platform along the S-curve is relevant only to the technology solution itself.

3.1.1.2 The emergence of a dominant design

The pre and post-dominant design phases in a platform's Lifecycle are mostly relevant with the Market. In the pre-dominant design phase, there are several competing designs in the market, and user expectations are not yet defined. This is

a phase of high uncertainty, where Real Options have the most value. Real Options are connected the most with Strategy and will be more thoroughly explained in the relevant section. In the post-dominant design phase, user expectations are set on a gold standard and, at this phase, all different designs are variations of the dominant one. This does not mean that all designs use the same technology, but mostly that user expectations are now set on a required set of features. At this phase, differentiation is heavily dependent on complementary functionality and the competition also shifts to prices [1].

3.1.1.3 Diffusion amongst users

The last dimension of a platform's Lifecycle is represented by the Diffusion Curve, which depicts the adoption of the platform among end-users or the app developers. The Diffusion Curve is divided into five phases associated with a percentage of its total adopters: Geeks (3%), Early Adopters (12%), Early Majority (35%), Late Majority (35%) and Laggards (15%). Strategic choices and governance policies (e.g. pricing) are affected by the phase of the platform along the Diffusion curve among end-users. An example of this is Amazon's Kindle, where at the early stages of the diffusion curve, the pricing end of the end-users was subsidized, to attract book publishers [1]. Once the platform reaches the Early Majority stage, the subsidies can be gradually reduced.

3.1.2 Lifecycle and Strategy

The correlation of Lifecycle and Strategy has also been examined by [36]. Even though only the S-curve is explicitly examined and connected to the Business Model of the platform, the authors also implicitly examine the market's uncertainty, as it was discovered to affect the platform's strategy. Certainty in the platform's market seems to be relevant to the pre and post-dominant design phases found in [1].

According to [36], companies have different strategic profiles depending on their current stage in the S-curve. In the early stages, companies adopt an Open Platform Strategy where contribution in technology creation by the third-party developers is encouraged and supported in a great degree. Adoption of the platform by as much app developers as possible is desired, as minimal screening is performed and the platform is also open to latecomers [36]. This strategic profile is mostly adopted in immature markets, where firms are in the introduction and ascent phases [36]. The same notion is found in [1], where embedding Real Options in the architecture, providing support to the app developers and adopting lucrative governance policies are highly valuable strategies in immature industries where rapid innovation is critical.

The most common strategic profile in the maturity phase, according to [36], is the profile of the dominating platform. Dominating platform strategy aims to control the direction of the ecosystem's development and scale up the product volumes. The keystone firm makes more use of screening when it comes to its partners, it does not provide access on the platform core to the app developers but encourages them to contribute to the platform's supplementary part [36]. Most commonly, this strategic profile is adopted when product design is finalized, and there is low uncertainty in the market. However, mature firms have been seen to

adopt an open platform strategic profile when the industry is still in an immature phase.

In the latest stages of a platform's Lifecycle along the S-curve, keystone firms are on the look for new markets to penetrate [1] [36]. The Opportunistic strategic profile described in [36], is usually found in platform ecosystem's that have reached their decline phase. Platform owners of such ecosystems are actively seeking markets where their design is superior to the incumbent designs of the penetrated market [1]. Therefore, they target markets in their mature stage [1] [36], where their technological solution can deliver value to the customers, with some additional development effort [36] [1]. A platform that adopts an opportunistic strategic profile, is on the look for opportunities to "leapfrog" to the next S-curve. To achieve this, the platform owner must embrace the disruptive technology that set the platform into its "Decline" phase and use it as a foundation to build new functionality. This requires the platform to be flexible, which can be achieved by embedding real options into it. Then, the platform can evolve and adapt by either exercising growth options embedded into the architecture, a move called Mutation, or by enveloping adjacent markets [1]. Mutation and Envelopment will be more thoroughly explained in the following section, which is concerned with Strategy.

3.1.3 The Lifecycle of PS Company and its effects on its ecosystem

The Lifecycle of PS could be described as follows: In the early stages, the proprietary platform of PS is introduced to both customer segments (C and SI). The percentage of its total adopters at this point is under 15% (Early Adopters and Geeks). At this point, the market is at the pre-dominant phase. Pre-dominant phase means more uncertainty and gives increased value to Real Options, affect both the Architecture and Structure of PS. Likewise, in the post-dominant phase, PS should take different paths when it comes to the same aspects. As the platform goes through its Lifecycle, PS must manage its Architecture, Governance and Structure to fit its Strategic goals. At maturity, where usually a dominant design has already set in. PS must manage the main bulk of adopters and create revenues through economies of scale, while at decline, PS must exercise growth options, renew the market or envelop adjacent ones to keep its business model returning revenues. To summarize, PS has different goals in each stage of its Lifecycle, and so it follows a different strategy.

3.2 Strategy

A platform's strategy dictates its governance, structure and in extent its architecture. It also changes according to its Lifecycle stage. We have identified several approaches to ecosystem strategy in the literature [37] [36] [1] [38]. Most of them claim that the keystone firm or platform owner is the key decision maker in an ecosystem. In [38], the authors present a model which classifies the characteristics of a SECO, and claim that a keystone firm might find it helpful to make more informed decisions concerning its strategy if it collects data relevant to these characteristics. They also conducted a case study as evaluation to their model, and the case company was satisfied with the model and made more informed strategic

decisions. However, the results of these decisions remained to be seen when the article was written.

Both [36] and [1] clearly connect a platform's strategy with its Lifecycle stages, and both agree on the general strategy that a platform should have in each of them. There are two main differences between these sources. The first main difference is that [36] analyses a platform's strategy using a three-view model while [1] analyses strategy based on evolutionary metrics and architecture-governance alignment. The second difference is that [36] focuses only on the S-curve of the platform's Lifecycle, while [1] also includes the pre and post-dominant phases of the industry and the Technology Diffusion curve.

3.2.1 Real Options

The notion of Real Options is much more relevant to Strategy than any other dimension defined in this paper. Real Options is a way of thinking that controls how a project can be structured to maximize value and limit potential losses, in an uncertain future [1]. It is an investment strategy, which has the purpose to provide flexibility in a project. Flexibility in this case means that the product owner (the keystone firm in our case) has the ability to quickly react on external stimuli [35]. This can be achieved by investing early in innovations that might prove useful in the future and by taking specific implementation decisions in a certain way [1]. When a project has acquired this type of flexibility from a specific real option, it is said that it has the real option embedded in it [35] [1]. The Real Options that are of interest in the scope of this paper are strategic or growth options. According to [1] "a strategic option (also known as a growth option) refers to future flexibility to use the project as a foundation or stepping-stone for yet-unimagined follow-on projects through further investment."

Real Options have the most value in volatile environments [1]. Volatility could come from the market (e.g unknown end-user needs and expectations or rival companies) or from technology (e.g. Immaturity of technology or unpredictable evolutionary trajectory) [1]. According to literature, an ecosystem's environment is at its most volatile stage at the early stages of its Lifecycle (pre-dominant phase and early stages in the S-curve) [1] [32].

3.2.2 Network Effects

An important notion which is more related to a Platform's Business Model is the notion of Network Effects. Network effects are created in a platform's ecosystem, when the accumulation of adopters in any of both customer segments (App developers or end-users) increases the value of the platform. Network effects might be same-side or cross-side, with the first meaning that a big number of adopters in one customer segment increases the platforms value in the same customer segment, while the latter means that it increases value in the other customer segment. An example of same-side network effects, is the popularity and attractiveness to the end users of a messenger or social network application. Its value increases exponentially when the number of its users is increased. An example of cross-side network effects, comes from the contemporary mobile platforms like iOS or

Android. A big number of app developers makes the platform more attractive to end-users and vice-versa. Strong network effects are created once the platform reaches a critical mass of adopters in both sides, when the effects become self-sustaining. Network effects are very important for a platform and can be the means to achieve dominance and a big, sustainable market share.

3.2.3 Evolutionary Metrics

A keystone firm can use evolutionary metrics to steer a platform's evolution, attain better information from its environment and make better informed decisions regarding tradeoffs in the design [1]. Evolutionary metrics are concerned with emergent properties of the platform, the apps or the whole ecosystem, which affect the ecosystem's survival rather than just its flexibility. The terms 'evolutionary metrics' and 'evolutionary attributes or properties' are used interchangeably in this paper. Of course, evolutionary metrics are used to track evolutionary properties, hence the name, but the exact metrics are outside the scope of this paper.

The evolutionary metrics that a keystone firm should look out for are divided into three timeframes. Short-term metrics are scalability, composability and resilience. Medium-term metrics are stickiness, platform synergy and plasticity, and Long-term metrics are envelopment, mutation and durability [1]. These metrics are tangible and can be measured in several ways. Automation in metrics measurement and monitoring, however, is of great value to the ecosystem. The author in [1] claims that an ecosystem's evolution should be orchestrated in respect to these three different timeframes, and proposes ways to align architecture and governance to optimize for each one of them. The keystone firm should focus on the short and medium-term metrics without losing sight of the long-term. In this thesis, we will adopt this framework to analyze a platform's strategy.

Concerning the relation of strategy with the architecture, all three metric groups have a direct or indirect effect on the architecture and vice versa.

3.2.3.1 Short-Term Metrics: Resilience, Scalability and Composability

The short-term evolutionary metrics are Resilience, Scalability and Composability. Resilience describes the tolerance of the ecosystem to failures within or outside it. Scalability is concerned with whether the platform is size-agnostic. Composability is concerned with the freedom of the platform owner to make changes within the platform with no negative effects to its apps.

3.2.3.2 Medium-Term Metrics: Stickiness, Platform Synergy and Plasticity

The medium-term evolutionary metrics are Stickiness, Synergy and Plasticity. Stickiness is concerned with how much is the platform used by its customers (app developers or end-users). Platform Synergy describes the degree on which the app developers develop an app specifically for the platform. Plasticity is the ability of the platform to add new functionality to cover needs that were not expected on design time.

3.2.3.3 Long-Term Metrics: Envelopment, Durability and Mutation

(a) Envelopment

Envelopment is the expansion of the platform either in adjacent markets (horizontal) or in the value chain (vertical). Most common envelopment moves are horizontal, where a platform implements the functionality of another product or service with which they share a customer base. Vertical envelopment moves can be upstream (when a platform includes in its core functionality that was traditionally provided by a supplier) or downstream (when a platform includes functionality provided by one of its app developers).

(b) Durability

Durability is the competitive toughness and endurance of the platform in the market. It is concerned with how much of its active user base it retains through the years, how well has it resisted against rival envelopment moves and in general it is an indicator of how well can the platform adapt and survive in a competitive market. Some ways to promote durability are:

- Amassing valuable and differentiating functionality before it can be imitated by rivals [1].
- Managing architectural corrosion by adding new APIs for new functionality and slimming down the platform's interfaces [36] [1].
- Identifying the Lifecycle stage of the ecosystem and understanding its consequences [1].
- Upstream Envelopment moves [1]

(c) Mutation

Mutation is the use of a platform's spin-off to a whole new market. In such situations, the keystone firm penetrates a new market with the technology of its platform, which acts as a disruptive design, and by also having a carryover of end-users to its new venture. In this situation, the platform owner uses its technology for a different application. Of course, identifying the correct market to penetrate needs dedicated organizational resources to scan mature markets and assess the competitiveness of the platform's technology in comparison to the competitors [1].

3.2.4 Strategy and Architecture

The relationship of architecture to the evolutionary attributes that were described above is direct in the short term and indirect in the long term. The long term emergent properties of a platform are dependent on the medium and short-term properties. The Short-Term properties of the platform are directly dependent on its architecture which should be complemented by Governance tools [1]. Modularity of the platform, and the existence of precisely documented and stable interfaces are the most important preconditions for all metrics. Even though the metrics are different, they all are satisfied by a modular architecture but in different ways.

(a) Architecture and Short-term metrics.

A modular architecture bolsters the platforms resilience by decoupling platform from apps and minimizing ripple effects that may happen because of maintenance, update or failure in one part of the system [34] [1]. One additional measure to improve resilience is to design for redundancy when it comes to external to the platform services. This, if applicable, can be achieved by using industry standards. Another way to do this would be to engage in upstream envelopment, a long-term strategic movement and develop the functionality offered by the external services in-house. An example of this is the in-house implementation of Cloud Services (iCloud) by Apple. [1]

To be scalable, a platform should standardize and explicitly define its interfaces and make sure that the apps communicate with the platform only through them. Explicitly defined standard interfaces have also been mentioned as a necessary requirement for SECOs [39] [3] [40] [1]. Composability also requires modularity and explicitly documented interfaces. However, there seems to be a trade off when it comes to internal modularity, between resilience and scalability. Even though a monolithic architecture within the platform is preferred for resilience, a modular architecture within the platform is necessary when it comes to scalability. That is because, it would be a wise choice to create bundles of functionality that are expected to scale upwards as modules within the platform [1]. However, internal monolithicness is preferred within the platform, if the platform's implementation is under the platform owner's control, because it is expected to reduce failure rates compared to an internally modular architecture and improve resilience. Nevertheless, a designer should provide the platform with the ability to scale downwards, in case the adoption of the platform goes worse than expected. Lowering fixed costs and, in extent, the threshold over which positive revenues can be achieved is a way to do this. For this reason, the platform's designer should create a platform with a minimal initial footprint rich in Real Options. A tenet describing this tactic is to "design for your dream audience but provision for the expected load" [1]. Real Options thinking, combined with a layered architecture [41] and the required Organizational Structure (Governance tools and capabilities) [30] [39] is one way to embed strategy into the architecture.

(b) Architecture and Medium-term metrics

Medium-term evolutionary properties are also affected by the architecture, but in a less direct way. Governance and Structure dimensions seem to play a bigger role in this timeframe, as well as the Lifecycle dimension. Concerning the architecture, modularity plays, once again, an important role, because it makes it easier for the app developers to develop complementary functionality, increasing Stickiness, and it also increases the flexibility of the platform, which directly affects Plasticity [1]. Making the development process easier for the app developers is further supported in literature [3] [40]. According to [40], app developers are influenced by the support that is offered from a platform. This relates directly to Stickiness which is positively affected by making it easier, cheaper and faster for the app developers to develop apps [1]. To reduce the costs for the app developers, and further bolster Stickiness, a platform owner should evolve the platform by retiring legacy functionality, adding new APIs, recognizing commodity functionality, proactively building on external innovations (Real Options thinking) and adding generic functionality by engaging to upstream envelopment moves [1] [39]. Apart from the architecture, Strategic Incompatibility with rival platforms is another way to increase stickiness, by increasing switching costs for the platform's customers [1].

Growing the end-user pool and ensuring access to it to the app developers is also very important [3] [1].

What affects Stickiness also directly affects Platform Synergy, but one additional rule that the platform owner should have in mind is to avoid downstream envelopment moves, except for extraordinary cases [1]. Downstream envelopment moves will make app developers more reluctant to adopt the platform, and in the long run may hurt the ecosystem. Platform owners that engage in downstream envelopment are described as "dominators" in [42], and it is argued that they are harmful to the Ecosystem in the long run.

(c) Architecture and Long-term metrics

We did not find any direct relation of long-term strategy metrics with the architecture, but there are causality and correlation relationships with the medium and short-term metrics [1]. So, we could say that long-term strategy metrics and emergent features of the ecosystem are indirectly related to the architecture. Also, the long-term evolutionary properties of the platform, are more closely related to the platform's Lifecycle.

Durability has a causal relationship with a platform's stickiness, which makes sense, as a big pool of app developers leads to a big number of customers, and strong network effects are one non-substitutable asset for a platform [1]. Furthermore, durability is a direct function of a platform's differentiating functionality. Valuable, rare, inimitable and non-substitutable functionality give a competitive advantage to the platform and should be constantly be accumulated [1]. The concept of durability can be found also in [39] where adding new functionality as new APIs and slimming down the platform as needed, by retiring legacy functionality, are considered two of the necessary tasks that a platform owner should have in mind. The same notion is found in [38], where it is stated that a platform's R&D is critical for ecosystem Health, because the app developers need innovations in the platform that can further support them to innovate on complementary products. Based on the previous points, one could say that Stickiness with the app developers and Synergy have a strong effect on a platform's durability. Decentralized governance also increases durability, as it provides the app developers with the freedom to react to their local markets [1].

Envelopment and mutation are two different ways for a platform to move to a new market, evolving the business model. The first one is concerned with entering adjacent markets by including new functionality and, thus, providing the ability to transition to another future core business. Spotting such opportunities and having the ability to act when they appear, involves dedicated units in the organizational structure, to watch adjacent mature markets and combine emergent technologies. It also, requires the platform owner to be very well tuned to the desires of its end-users. These can be achieved with a combination of organizational restructuring and Real Options thinking for the first, and by including input from the app developers in the platform's strategic decisions for the latter. The latter is based on the argument that app developers are closer to their niche markets and can provide useful insights concerning horizontal envelopment opportunities. Horizontal envelopment however will only extend a platform's maturity stage, and will not create a new business [1].

Another way of adding new functionality into a platform's core is by means of vertical envelopment, which can either be upstream or downstream in the value chain. Upstream envelopment, as explained before, is a way to also increase a

platform's durability [39] [1]. Downstream envelopment, is including the functionality of an app developer in the platform core, leaving the app developer out of business. The latter is strongly discouraged by the literature [42] [1], as it might discourage app developers from joining the platform and deprive it of its innovative potential.

Mutation is another way for a keystone firm to move to a new market, by leveraging its technology to penetrate it. However, in contrast with envelopment moves, the keystone firm breaks off completely of its business model by creating a spin-from the original platform and uses its technology in a way that was unprecedented at design time [1]. This is a way for a platform to escape a dying market, or a losing battle against a newer disruptive technology [36] [1]. By assessing the platform's assets, the keystone firm can identify new markets where its technology provides a great competitive advantage. Though not directly relevant with the architecture, mutation is an evolutionary metric which is promoted by the same governance structures that promote composability and durability. Furthermore, a platform's plasticity has a causal relationship with mutation [1].

3.2.5 What would PS do: Strategy

PS has different goals in different phases of its Lifecycle which dictate its Strategy. To succeed in its strategic goals, PS must take specific decisions concerning the Architecture, Governance and Structure of the Ecosystem. Adopting the approach of evolutionary metrics concerning Strategy, PS would make sure to be able to attain the long-term evolutionary attributes by making decisions on the short-term. For example, to be able to do envelopment moves, PS must pay close attention to feedback coming from cooperating SI who might have better insights on adjacent markets that could make good targets for a horizontal envelopment move. Furthermore, vertical envelopment moves would make SI's platform more durable in the market.

To achieve its long-term goals, PS would have to customize the dimensions of Architecture, Governance and Structure to satisfy the medium and short-term evolutionary metrics. A resilient and scalable platform with good composability, would promote plasticity, stickiness and synergy, making the platform compelling to SI and Customers. It would also give the platform the ability to adapt to a dynamic or changing environment, rebuff rival envelopment moves and maybe survive a Red-Queen Effect initiated by a rival platform by being able to adopt the disruptive (soon to be dominant) technology and making a successful leap to the next S-curve.

3.3 Governance

Ecosystem Governance is a foundational aspect on which the survivability and the realization of the ecosystem's strategic goals depends [32] [1] [43]. According to [1], Governance and Architecture are the two main factors which affect the evolutionary trajectory of a SECO. Both must be aligned to ensure sufficient support to Open Innovation and in extent survivability of the ecosystem. Most choices regarding this matter lie with the platform owner [1] [34]or coordinator

[44], who needs to have some type of control over the innovation process. Traditional formal governance mechanisms are not sufficient in the case of the SECOs [34] [3] [4] [1] and process control is advised against [1] [34].

Instead of the word "control", a much more appropriate word concerning governance is "orchestrate". Using non-coercive informal control mechanisms, the platform owner can guide the app developers to develop applications that promote the ecosystem's interests. To the best of our knowledge, SECO Governance is a subject that has not been directly and widely addressed in the literature. One example of a paper which refers to governance in a more organized fashion and introduces a Governance model is [44]. It presents governance tools and mechanisms that can be used by the keystone firm to preserve and improve Ecosystem Health as is introduced by [42]. In [42], three Key Performance Indicators concerning the health of a Business Ecosystem are defined. These KPIs are Robustness, Productivity and Niche Creation. In this thesis, however, we will adopt the Governance Model described by [1]. According to [1], Governance has three dimensions, all of which should be aligned with the ecosystem's architecture, strategic goals and Lifecycle stage. These dimensions and their influence to the architecture and vice versa, are presented in the following subsections:

3.3.1 Decision Rights Partitioning

This Governance dimension is concerned with the decisions that the stakeholders are entitled to make, concerning what the respective subsystems should do and how it should do it. In a modular architecture, the decision rights should be split exactly as the respective modules are split in the ecosystem architecture. This is called the mirroring principle [1].

Furthermore, a second criterion concerning partition of decision rights is that decision rights should lie with the stakeholder possessing the most knowledge for that decision. Specifically, decision rights are classified into two types: Strategic and Implementation. Strategic decision rights are concerned with what should a subsystem do, while Implementation decision rights are concerned with how a subsystem should do what is decided by the Strategic decisions [1].

3.3.1.1 Application Strategic Decision Rights

To support Open Innovation, and give the app developers the necessary freedom to come up with new ideas that will address the respective market niches in which they operate, Strategic decision rights concerning the apps are fully decentralized to the app developers. This also ensures that the app developers can act and adapt as fast as they need to in a volatile market, or resist envelopment attacks by rival platforms [1].

3.3.1.2 Application Implementation Decision Rights

Application Implementation rights lie also with the app developers, but it would be beneficial for the ecosystem if the app developers received some input from the platform owner. The reason for this, is that the platform owner can provide crucial input concerning the functionalities that are offered by the platform's technology. This can help the app developers to make use of the platforms features to the full extent while it would also make integration easier.

3.3.1.3 Platform Strategic Decision Rights

Platform Strategic decision rights are concerned with what the platform should do and set the direction concerning the platform's evolution. Even though the bulk of this knowledge lies with the platform owner, input from the app developers should also be included. The reason for this is that app developer input contains crucial information about a platform's users' needs [1]. This includes both app developers and end-users. App developers are the stakeholders that could suggest useful functionalities for the platform to include in the future, and they could also provide directions to the platform owner concerning how the platform's interfaces should evolve. The predictability of evolution of the platform's interfaces has been brought up as a challenge for SECOs also by [39]. Furthermore, app developers are closer to the end-users of the platform, which means that they get a more accurate pulse concerning end-user needs, trends and relevant useful information.

3.3.1.4 Platform Implementation Decision Rights

A platform's implementation rights however lie with the platform owner, because it is the platform owner that knows best about the technology of the platform and should decide about features, UI etc. Furthermore, the centralization of the platform's implementation decision rights gives the platform owner control over its architecture and, in extent, control over its evolutionary trajectory.

3.3.1.5 Summary

To summarize, partitioning of Decision rights in the ecosystem is enabled by architecture and it is important for several evolutionary attributes of the Ecosystem as it provides it with the flexibility to evolve and adapt as needed. Furthermore, it is also crucial in bolstering Open Innovation as it provides the app developers the necessary freedom to co-create value for the end-users, making the platform more popular and competitive.

3.3.2 Control Portfolio

This Governance Dimension is concerned with the control that the platform owner can exert to the app developers. The need for coordination mechanisms across an ecosystem has been widely identified in the literature [30] [34] [43] [1] [3], however more specific tools of control (coercive or not) are much more harder to find. The highly distributed nature of Open Innovation in Software Ecosystems, makes traditional governance mechanisms undesirable [45].

Concerning proprietary platforms, split revenues can partially accomplice goal convergence in the ecosystem between the platform owner and the app developers [1]. Apart from that, a general rule of the thumb when it comes to control policies for the platform owner, is to plan a simple control portfolio optimized for minimal cost to the platform owner and the app developers [1]. Formal control mechanisms like Gatekeeping and Process Control might be of use in this case but

they should be complemented with and informal control mechanism, namely Relational Control and they should be applied in a minimal manner [34] [1] as it will be explained later.

(a) Relational Control

Relational Control is non-coercive to the app developers and it involves the communication of shared values and the cultivation of an organizational culture across the ecosystem's participants, from the keystone firm [1]. Relational control seems to be much more effective in Open Innovation collaborations [46] [1] [47] but it is not sufficient to ensure smooth integration. Non-profit open source communities can demonstrate the power of a common set of values across an ecosystem, with Linux being a powerful example [47]. A platform owner can foster an ecosystem-wide culture by setting examples through its own actions, reinforcing a common identity among the ecosystem's participants, organizing socialization opportunities for the app developers and pre-screening app developers which works as a way to preserve the culture [45] [30] [1].

However, relational control is difficult to maintain or cultivate in an ecosystem with a high turnover of participants [1]. Therefore, some screening of possible app developers that want to adopt the platform is advisable [1] [44] [30]. The correct approach to this problem is complementing Relational Control with a formal control mechanism. Process Control, as a main control mechanism, has been advised against in literature [34] [1], while a formal process of screening, called Gatekeeping, has been mentioned as a way to hinder harmful to the ecosystem/platform applications from disrupting the ecosystem [34] [1] [30].

(b) Gatekeeping

Gatekeeping involves the use of explicitly set criteria about which applications are accepted to be integrated with the platform. Transparency, of course, in Gatekeeping is crucial as the trust of the app developers, which in turn affects some evolutionary metrics, is affected. Gatekeeping should be used in a way that does not hinder innovation from the app developers. A way to do this is for the platform owner to explicitly communicate to the app developers what an app should not do, and then use gatekeeping to screen apps with these criteria in mind. An example of the platform owner clearly communicating what the app developers should not do, is Apple's prohibition of apps that duplicate the native functionality of iOS. Gatekeeping is essential in fully extracting value out of a modular architecture [1]. Subsequently, a modular architecture is essential to coordination in the ecosystem. Process control can increase the value of Gatekeeping if used with the intention to help app developers pass the Gatekeeping tests [1].

(c) Process Control

Process Control is relevant with how much the platform owner relies on incentives and counter-incentives towards app developers concerning if they follow prescribed methods of development and procedures to achieve the desired outcomes defined by the platform owner. Process Control requires from the platform owner to have the knowledge to mandate methods to the app developers and the resources to

monitor them. As a traditional control mechanism, it is advised against as the primary control mechanism, because the platform owner not always has the necessary knowledge concerning the app developers' day-to-day work. Furthermore, it might restrict the app developer's freedom as it is contradictory to the decentralized implementation decision rights partitioning that is suggested in literature. Therefore, the only option where the platform owner is suggested to use process control, is to help the app developers pass the Gatekeeping checks [1].

These formal and informal control mechanisms can and should be complemented with organizational capabilities [1] [30], which will be presented in the section relevant to Structure.

3.3.3 Pricing Policies

This Governance dimension is also closely tied to the Business Model of the platform and its Lifecycle stage [1] [31]. The platforms Lifecycle stage, its Business Model and the Architecture must be aligned because pricing decisions can create incentives for the app developers to contribute to the platform, resulting in a competitive advantage for the platform [1] [30].

According to [1], there are five considerations when it comes to pricing policies: Pricing Symmetry, Subsidy-side, Access/Usage Fees, whether to use a Sliding Scale and the App pricing model. The first two considerations are closely tied to the keystone firm's Business Model and its Lifecycle stage. The rest are also related to the Architecture and, consequently, of more interest to this thesis.

Pricing symmetry is concerned with whether the platform owner will use positive pricing to both sides or not. Platforms that have reached critical mass in one side and then open up to a second side can make money from both sides. For example, a product line with a big pool of end users has a lot of possibilities when transcending to a platform, that the other side opened (app developers) will be willing to pay to adopt the platform. The only other case where Symmetric pricing might work is if the platform owner is an early mover with a dominating design/product [1]. In any other case the platform owner should subsidize one side.

The second consideration is concerned with which side to subsidize. Usually, in the early stages of a platform, and in a pre-dominant design phase of the Industry, the platform owner should choose to subsidize the App developers, as rapid innovation is crucial [1].

However, the ecosystem's architecture, and especially an app's micro-architecture plays a significant role in pricing when it comes to usage fees. If an app's microarchitecture is heavily reliant on the platform's native services and these resources are also not scalable, then usage fees can be imposed. However, this would impose monitoring costs on the platform owner and would signal that the architecture lacks scalability in the specific module. In this way, the platform owner can identify weaknesses in the platform's modularity [1]. A sliding scale in pricing is also advisable if platform's scalability is low.

A platform architecture seems also to constrain the viability of the apps' business models. An app's microarchitecture is also concerned with what functionality will be executed at the app level and which functionality will be executed at the platform level, using the platform's API's [1].

So, it seems that a platform's architecture influences the platform's pricing policies but pricing policies do not affect the architecture.

3.3.4 What would PS do: Governance

To steer the ecosystem's evolution, PS must use more subtle Governance techniques than the ones usually found in traditional enterprises. Since, strict process control mechanisms are useless in this case, PS must find ways to guide the SI to the direction which benefits PS without restricting them, creating a minimal but effective control portfolio.

To make its platform lucrative enough to the SI, and to provide incentives concerning the microarchitecture of their apps and their toll on the platform, PS should adjust its pricing policies. Furthermore, to ensure sufficient coordination between cooperating SI and PS, and promote seamless integration of the respective products (platform and apps), decision rights concerning what the platform or an app will do and how, should be rightly divided.

For example, PS will always conform to the Mirroring Principle and have the SI be responsible for what will their complementary software do. However, it might provide a little input concerning the implementation, as this might make it easier to achieve good integration. Furthermore, PS might want to warn the SI about something that they should not do when implementing additional functionality to the platform. Using Gatekeeping to keep harmful software outside the platform and a mild process control as a support tool to the SI to pass the Gatekeeping checks are two possible control mechanisms for this purpose.

Also, at different stages of its Lifecycle, and for different reasons, PS might adopt different pricing policies. To guide SI to not set up their micro-architectures in way that takes a heavy toll on PS's resources, PS might introduce usage fees. Furthermore, to make it past the stage of the Geeks and Early Adopters in the diffusion curve, PS might provide some of its functionality for free.

3.4 Structure

When an organization decides to adopt an Open Innovation model, and sets up a platform, it also needs to go through organizational restructuring [30] [3] [1], and create new business units and organizational capabilities. To support Open Innovation, the platform owner should create a strong championship [30] [1], provide incentives to promote Open Innovation effort and create cross functional teams and Open Innovation business units [30] [1], while also promoting an organizational culture in favor of Open Innovation [30] [46] [1].

A platform owner should also invest in technology tools to provide support to the app developers [3] [30] [43] [40] [1] as this has shown to increase the adoption of the platform from the app developers [40] while making it easier for them to contribute to the ecosystem with complementary to the platform functionality [3] [1].

One aspect of Structure that greatly affects the platform's architecture is Real Options Thinking. As written before, embedded Real Options in the Architecture provide the platform with the necessary flexibility to adapt to a volatile environment. Creating alternative options for a platform which can be utilized when

needed needs dedicated R&D resources, charged with the task of finding and combining external to the ecosystem innovations, assessing their usefulness and either including them in the architecture or discarding them [1].

Regarding the architecture, Real Options could be combined with the Three-Layer Product Model [41], as they seem to be the equivalent of the experimental functionality layer. The Three-Layer Model will be explained in more detail in the Architecture section. Real Options are much more valuable in volatile markets, whether the volatility comes from the supply side and is technical or whether it comes from the market side and is relevant to the end-user demands. So, Real Options value is increased in the pre-dominant design phase of the platforms Lifecycle, where end-user needs are not yet clearly defined [1] [36]. Real Options might also be realizable when the platform reaches maturity in the S-curve, and strategic moves like mutation or envelopment might be necessary for the survival of the platform [36] [1].

Thus, Real Options affect a platforms architecture in the following way: In markets of high uncertainty, when Real Options have the most value, modularity within the platform is preferred to include experimental functionality as described in [41]. Furthermore, in the early stages of a platform's Lifecycle, the architecture of the platform should have a minimal footprint (that is including only the bare necessary functionality) but should be rich in Real Options. This reduces fixed costs and makes the platform downwards scalable. In markets with low uncertainty (a dominant design has emerged and user expectations are known) the platform owner should switch to Open Source in functionality that lies in the commodity layer to reduce ownership costs. Monolithicness might be desired within the platform at this phase because it can outperform modular architectures due to the absence of communication overheads between the modules. An exception to this is bundles of functionality that should scale. Thus, after the late majority stage in the platform's diffusion curve, the platform's architecture could become monolithic as no more users are expected to adopt the platform while the platform owner should mutate the platform by creating a spin-off and penetrate another market [1].

What would PS do: Structure

At the point where PS would set up an ecosystem around its platform, it should create or refactor Organizational Capabilities to support it. R&D would have different goals than before, including evolving the Architecture by embedding highly valuable to the SI functionality, instead of trying to create value for lots of different niches. PS should also spend some resources to tools and technologies which support the SI to create complementary functionality to the platform. However, PS would allocate its resources differently depending on its Lifecycle stage. In the early stages, where Real Options have the most value, the bulk of R&D resources should go into assessing and embedding Real Options in the platform's Architecture. To extend its maturity phase, PS would have its R&D units identify good horizontal envelopment targets and shift functionality from the experimental to the differentiating layer, and from the differentiating to the commoditized layer. In the decline phase, PS would have its R&D units engaged in process innovation instead of content, and also check for mature markets to penetrate with a mutation move.

3.5 Software Architecture

Software Architecture is an aspect that plays a pivotal role in a Software Ecosystem. It is a key enabler for Governance of the ecosystem by the keystone firm, and it also greatly affects the short-term, medium term and long term strategic goals of the Ecosystem, and in extent its survivability [1] [40]. Furthermore, the ecosystem's architecture affects and is affected by the organizational structure of the keystone firm.

Specifically, an ecosystem's survivability is directly affected by the extent on which its subsystems can evolve and adapt to the rapidly changing environment of the ecosystem [1]. This means that both apps and platform must evolve over time to keep the ecosystem competitive enough to survive in the market. Once again, the platform owner must find a balance between two extremes. First, evolving too aggressively would make the platform unattractive to app developers, as it would not leave them enough space to innovate and create value. On the other hand, a platform that does not evolve will eventually become commoditized and, thus, will not offer differentiating functionality, over which the app developers can build. This will also result in an unattractive platform to the app developers, and in extend to the end users [39].

By using evolutionary metrics, the platform owner can monitor and affect short, medium and long-term attributes of the ecosystem, which are directly related to its evolvability. However, the platform owner should always remember that the architecture of the ecosystem plays a critical role concerning these evolutionary attributes and that the platform owner is the key decision maker concerning this matter.

3.5.1 Architecture and Modularity

A major enemy of an ecosystem's longevity is complexity [41] [39] [1] [34] [43] which can erode a platform's architecture, create the effect of lock-step evolution (where unwanted dependencies between apps or the platform restrain the evolution of either the platform or the apps) and several other unwanted effects.

A key property of an ecosystem is the modularity of its architecture [39] [34] [3] [1]. Specifically, the platform and the apps should only communicate through interfaces, which should be precisely documented [39] [40] [1], stable [39] [40] [1] and that evolve in a predictable fashion [39]. Modularity can also exist within the platform or the apps, if the relevant stakeholder desires to [1]. However, perfect modularity is outperformed by monolithic architectures due to the communication overhead between modules and adds to the cost structure during the implementation. So, in general, monolithicness is desired within applications and the platform, unless rapid innovation is more important than performance which could be the case in immature platforms or new and dynamic markets [1]. Furthermore, two arguments have been made in favor of modularity within the platform.

3.5.1.1 The 3 Layer Product Model

First, to further promote the evolvability of a platform, a model has been introduced by [41], where the platform is designed in a three-layered architecture, based on the

3LPM framework. According to [41] a platform is comprised of the Commodity Functionality Layer, the Differentiating Functionality Layer and the Innovation and Experimentation Functionality Layer. The Commodity layer includes functionality that the customer takes for granted and is does not differentiate the platform from its competitors. The platform owner should optimize for cost in this layer. The Differentiating layer includes all functionality that gives the platform a competitive advantage, and it should be optimized for maximum customer value creation. The Innovation and Experimentation layer is concerned with the future functionality of the platform. The main focus of the organization's R&D is on the activities happening in this layer. Furthermore, the app developers are also operating in this layer. These layers can evolve and release updated versions independent of each other, making the internal architecture of the platform modular.

[41] has also defined two interfaces through which functionality is relocated within the platform. The first one is the Commoditizing Transition interface, through which Differentiating functionality which has lost its competitive advantage and has become a commodity is transferred to the Commodity Functionality Layer, where it could also be replaced with open source software to minimize cost. The second one is the New Product Interface, where experimental functionality is relocated to the Differentiating Functionality Layer and adds further value to the platform for its customers.

In the same spirit, [1] proposes keeping highly reusable and generic functionality in the platform while letting experimental and uncertain functionality to the app developers. Furthermore, through the notion of Real Options Thinking, the authors claim that managers should always spend some resources in monitoring external to the ecosystem innovations and thinking of ways to combine them even if they seem absurd [1]. This means that some R&D resources of the platform owner should be spent in discovering future functionality that might be of great value to the app developers, assessing its value and incorporating it to the platform. A platform might also add new functionalities via envelopment, a strategic concept that will be discussed later.

One critical rule when it comes to adding functionality to the platform is exposing the new functionality by adding new APIs and not modifying the existing ones, thus achieving interface stability. In this way, backwards compatibility with the applications is not broken. This might result in a bloated platform interface and the architect should proactively plan for which APIs to remove from the platform, in order to delay its erosion [39] [1].

Also, a platform owner should avoid adding to the platform functionalities created by the app developers according to [1], unless some specific conditions are met. This decision is relevant Envelopment and Platform Stickiness (to the app developers).

The second argument in favor of modularity within the platform is that batches of functionality that are expected to scale upwards, should be separated in their own modules [1]. The result of this is a decrease in complexity, as the ripple effects of scaling upwards will be limited to the relevant modules.

Also, to promote the resilience of a platform, it is advisable to the platform designer to use industry standards when it comes to the use of services external to the ecosystem. In that way, the platform's redundancy increases, and functionality which relies on external services becomes more resistant to any possible failure of them [1].

In general, the most critical rule for the survivability of the ecosystem concerning modularity is that the platform and the apps should be decoupled and their communication should be limited to stable and thoroughly expressive interfaces.

3.5.2 What would PS do: Architecture

PS wants to set up its architecture in a way that supports its Governance policies. By decoupling the platform from the software developed by the SI and limiting communication to the platform's interfaces as much as possible, PS would create a good foundation for its Governance policies. In this way, SI have the freedom to create value for their respective niches and PS can still steer the evolution of the whole ecosystem in a non-coercive manner. Stable and expressive interfaces (APIs) are critical. The more well documented a platform API is, the less coordination effort will PS have to make. Ideally, the only communication between PS and SI would be through the platform's interfaces.

Regarding the platform's internal modularity, PS would choose a modular architecture in most of cases. There is a tradeoff between performance and flexibility as monolithic architectures tend to perform better than modular ones, and PS would want to take advantage of this fact in any case possible. This might be possible in the decline phases of the PS platform, where the industry has much less uncertainty in the market. However, at the post-dominant stage of the market, minimizing cost is also a priority, and a way to achieve this would be to switch commoditizing functionalities to Open Source. This might still require some modularity from the platform's architecture.

Adopting the 3LPM framework in its architecture is a good idea for PS, but with one more addition. PS should also separate functionalities that should be scalable in their respective modules. This will minimize complexity along the platform's Lifecycle.

Furthermore, to promote resilience, PS should use Industry Standards in the external services that it consumes.

4. Decision Support Model

In this chapter, the result of our study, the Decision Support Model (DSM), is presented. A Decision Support Model is a Conceptual Model which supports the decision maker by presenting the decisions to be made, their context, the choices given to the decision maker and a suggestion on which one is the better choice. The DSM is expected to be used by the Platform Owner (as the key decision maker) in the following way: Initially, the platform owner should identify the current stage in the platform's lifecycle. The platform owner can do this by comparing the current situation of the keystone firm (and the platform) vis-à-vis with the three Lifecycle dimensions as described in chapter 2. After this step, the platform owner should read the section Lifecycle Stages (4.1) below, to get advice concerning the Strategy of the keystone firm. Since Lifecycle governs the platform owner's Strategy, proper information feeds concerning the internal and external environment of the Keystone Firm and its Ecosystem are a necessity. After this stage the DSM provides support to the decision maker on how to align the three pillars of the Software Ecosystem (Architecture, Governance and Structure) with the Keystone Firm's strategic goals. The DSM is structured in the following manner: The first section presents the Lifecycle stages of the platform ecosystem and the optimal strategy, according to literature, that the platform owner should follow. Then, in the following sections, the decisions that need to be made are presented, described and supported as defined above, for each of the software ecosystem's dimensions. At the end of this chapter, an overview of the DSM (tables 11, 12 and 13) is shown to improve its usability followed by a diagram which connects the three Lifecycle dimensions of the DSM (figure 3).

4.1. Lifecycle Stages:

4.1.1. Lifecycle: Early Stages

Context
The industry is in a pre-dominant phase. This means that there are several competing designs and end-user expectations and needs are not defined yet. Thus, market volatility is high.

Strategy

- The goal is to create strong Network Effects (3.2.2) and come up as the owner of the dominant design and/or technology. Therefore, rapid Innovation is a priority and the platform should be as lucrative as possible to the app developers.
- Upstream and horizontal Envelopment (3.2.3.3-a) moves should be made if possible provided they offer valuable differentiation towards app developers
- Strategic Incompatibility with rival platforms is advised at this stage.

4.1.2. Lifecycle: Maturity/ Authority Stage

Context
At this Lifecycle stage, a dominant design has emerged, and user needs and expectations are known. Market volatility is not that high at this phase and product volumes and technology adoption among the end-users are rising sharply. Platforms at this stage, aim to increase product volumes and create revenues through economies of scale.

Strategy

- The goal is to be dominant in the focal market, to expand to other markets by horizontal envelopment (3.2.3.3-a) and drive up the product volumes.
- Differentiation happens on the app level, so a big number of App Developers is still desired, to cover more market niches and provide better feedback for envelopment attacks.
- If the platform is dominant, strategic incompatibility is advised. If the company is not dominant, one way compatibility with the dominant platform is advisable.

4.1.3. Lifecycle: Decline Stage

Context:

The industry is at a post-dominant phase, and the rate of Innovation concerning content is slowing down at a fast rate. As innovative breakthroughs concerning content are becoming increasingly difficult to achieve, platforms start to also compete in process Innovation which is more focused on delivering the same value with less cost

Strategy:

- The goal is to maintain the user base, and mutate to another market where the platform technology can be disruptive.
- Horizontal and Vertical envelopment moves are advised, to promote the platform's durability.
- By assessing the platform's assets vis a vis with the target market's incumbent dominant technology, good mutation targets can be identified.
- The platform owner should look for opportunities to engage in mutation moves (3.2.3.3-c). This involves penetrating mature markets using a spin-off of the platform as the disruptive technology.
- Two-way compatibility is also possible.

4.2 Decisions

4.2.1 Architecture

Goal:
The goals that the Platform Owner should strive to achieve concerning the ecosystem's Architecture are presented below for each of the platform's Lifecycle Stages:

i. Lifecycle: Early Stages

To create a design that will be flexible enough to adapt in the highly volatile market, but at the same time demonstrate enough stability to be trustworthy to third-party developers.

ii. Lifecycle: Maturity Stage

To have a platform with a stable core which offers differentiating and valuable functionalities to the app developers. Also, the platform should have the ability to support a great number of app developers, who are expected to evolve their apps at any rate. Furthermore, the platform owner should avoid a bloated platform interface without breaking backwards compatibility with any app.

iii. Lifecycle: Decline Stage

To minimize the effects of aging in the platform's architecture, slim down as possible the platform's API Interface and to reduce owner costs in commodity functionalities.

Decisions:

Concerning Architecture, the platform owner will have to make the following decisions

1. Modular or Monolithic?

The platform owner can choose between a monolithic architecture or a modular architecture with apps and platform decoupled. Furthermore, the platform could be modular internally. When a platform is modular, it can also have Real Options embedded in it. This means that at any point in time the platform owner can make an investment on the platform, and add in the design possibilities to use a variety of different technologies, which in turn might make the keystone firm able to grasp an opportunity or adapt to unexpected change. Owners of Cyber Physical Platforms should have in mind that scalability is of vital importance to the survivability of the platform given the data intensive nature of the domain. The platform in this case should be able to scale upwards and downwards with no restriction introduced by scalability requirements. This further suggests that a modular architecture with scalable functionality encapsulated in its own module might be the only viable choice that owners of Cyber Physical Ecosystem Platforms have.

Monolithic Architecture:

Pros: Greater Performance, Faster time to market
Cons: Increased Complexity with the tendency to increase throughout the whole Lifecycle of the platform. Bad Maintainability (Complexity). Bad Changeability (Complexity), High Integration effort

Modular Architecture:
Pros: Good Flexibility, Minimal Integration Effort, Good Maintainability, Minimal Complexity with minimized tendency to increase,
Cons: Higher upfront costs, Worse Performance than Monolithic architectures because of the communication overhead between APIs.

Suggested Action:

Concerning modularity, the Platform Owner is suggested to take the following actions for the different Lifecycle Stages:

i. Lifecycle: Early Stages

- Modular Architecture with apps and platform decoupled. Furthermore, internal modularity in the platform is advised in the following way: Scalable functionality should be in its own module. Experimental functionality should be decoupled from the platform's core and differentiating functionality.
- The platform's core functionality should have a minimal footprint (it includes only the bare minimum)
- New functionality should be added to the platform core by adding new APIs

ii. Lifecycle: Maturity Stage

- Modular Architecture with apps and platform decoupled. Furthermore, internal modularity in the platform is advised in the following way: Scalable functionality should be in its own module.
- New functionality should be added to the platform core by adding new APIs
- Experimental functionality should be decoupled from the platform's core and differentiating functionality. Commoditizing functionality should be decoupled in its own module, and optimized for cost. Towards this end, switching to Open Source Software is a way advised in the literature.

iii. Lifecycle: Decline Stage

- Modular Architecture with apps and platform decoupled. The 3LPM model is once again advisable.
- New functionality should be added to the platform core by adding new APIs
- The platform's interfaces should be precisely documented, stable and evolve in a predictable manner.
- External Services used by the platform should be based on Industrial Standards
- Use Open Source Software (OSS) for commoditized functionality.

2. Dynamic or Stable APIs?

The platform owner can either choose to update the interfaces of the platform during changes or addition of new functionality or to never change the interfaces and instead add new APIs.

Dynamic APIs
Pros: Lean interface profile.
Cons: Bad backwards compatibility, Negative effect on attractiveness to App developers', Increased Complexity due to the greater impact of platform changes to the apps.

Stable APIs
Pros: Decreased Complexity, Increased Attractiveness to App developers, Maintains Backwards Compatibility
Cons: Bloated platform Interface with legacy APIs.

Suggested Action:

Concerning APIs, the Platform Owner is suggested to take the following actions for the different Lifecycle Stages

i. Lifecycle: Early Stages

- The platform's interfaces should be precisely documented, stable and evolve in a predictable manner. Thus, New functionality should be added to the platform core by adding new APIs
- External Services used by the platform should be based on Industrial Standards

ii. Lifecycle: Maturity and Decline Stages

- The platform's interfaces should be precisely documented, stable and evolve in a predictable manner. Thus, New functionality should be added to the platform core by adding new APIs
- External Services used by the platform should be based on Industrial Standards
- The platform owner should slim down the platform's Interface by removing unused legacy API's.

4.2.2 Governance

Goal:

The goals that the Platform Owner should strive to achieve concerning the ecosystem's Governance are presented below for each of the platform's Lifecycle Stages:

i. Lifecycle: Early Stages and Maturity Stage

To provide enough freedom to third-party developers to innovate, while exerting influence over them to promote the ecosystem's interests by providing a context over which the app developers innovate.

ii. Lifecycle: Decline Stage

To provide enough freedom to third-party developers to innovate, while exerting influence over them to promote the ecosystem's interests by providing a context over which the app developers innovate. In this Lifecycle stage, one other goal of governance is to increase the input and incorporation of app developer feedback into the strategic decisions of the platform.

Decisions:

a. Decision Rights Partitioning

Decision rights are divided into two categories: Strategic and Implementation, for both Platform and Apps (3.3.1). Also, decision rights lie in a continuum between centralized and decentralized, and where each decision right lies is agreed between platform owner and app developers.

1. Centralized or Decentralized?

- **Platform Strategic:**

Platform Strategic Decision Rights are concerned with what the platform should do and specifically what functionalities should it have at its core.

Centralized:
Pros: Power to the platform owner to act for its product's best interest. Greater control over the platform's evolutionary trajectory. Stability in the ecosystem.
Cons: Extreme centralization might lead to a myopic view of the ecosystem's environment, and make it more difficult to set goals for the evolution of the platform.

Decentralized:
Pros: not specified in literature
Cons: Lack of stability. Lack of Convergent goals.

Suggested Action:
Concerning Platform Strategic Decision Rights the Platform Owner is advised to take the following actions:

i. Lifecycle: All Stages

Platform Strategic Decision Rights should lie with the platform owner, but not on the extreme. The platform owner should pay attention to input from the app developers, and act accordingly. That way, the platform owner is more attuned to the ecosystem's environment and can make more informed decisions which in turn will increase the platform's adoption by app developers and end-users alike. This promotes Durability (2.2.3.3-b). Furthermore, the platform owner is also more attuned with the app developers' needs which increases platform Stickiness with the app developers, and Synergy (2.2.3.2).

- **Platform Implementation:**

Platform Implementation Decision Rights are concerned with how the platform does what is should do.

Centralized:
Pros: Control over the architecture and in extent the app developers. The authority of making such decision lies with the stakeholders who possess the most knowledge for such decisions.
Cons: Not specified in literature.

Decentralized:
Pros: Not specified in literature
Cons: Lack of a stable core for the ecosystem. Lack of coordination among app developers.

Suggested Action
Concerning Platform Implementation Decision Rights the Platform Owner is advised to take the following actions:

i. Lifecycle: All stages

Platform Implementation Decision Rights should lie with the platform owner.

- **App Strategic:**

App Strategic Decision Rights are concerned with what the app should do and specifically what functionalities should it have.

Centralized:
Pros: Not specified in literature.
Cons: Restricts the Innovative potential of the app developers and it is counter-intuitive to the reasons a platform ecosystem is set.

Decentralized:
Pros: Maximizes Customization of the platform by using the innovative potential of several app developers.
Cons: not specified in literature

Suggested Action:
Concerning Application Strategic Decision Rights the Platform Owner is advised to take the following actions:

i. Lifecycle: All stages

App Strategic Decision Rights should lie with the app developers.

- **App Implementation:**

App Implementation Decision rights are concerned with how should a third-party application be implemented.

Centralized
Pros: Not specified in literature.
Cons: Not specified in literature.

Decentralized
Pros: The authority of making such decision lies with the stakeholders who possess the most knowledge for such decisions. App developers can upgrade and evolve their applications without constraints, as long as they use a modular micro-architecture through architectural encapsulation. Provides the app developers with the ability to multihome (support multiple platforms).
Cons: Extreme decentralization might deprive app developers from the potential to fully utilize the platform's unique functionalities and in extent make use of economies of scale.

Suggested Action:
Concerning Application Implementation Decision Rights the Platform Owner is advised to take the following actions:

i. Lifecycle: Early Stages and Maturity Stage

App Implementation Decision Rights should be decentralized but with some input from the Platform Owner. The input of the platform owner should aim to ensure interoperability with the platform and support the app developers to fully utilize the platform's functionalities.

ii. Lifecycle: Decline Stage

App Implementation Decision Rights should be decentralized but with some input from the Platform Owner. The input of the platform owner should aim to ensure interoperability with the platform and support the app developers to fully utilize the platform's functionalities. At this Lifecycle stage, however, App Implementation Decision Rights might be less Decentralized as a counterbalance to architectural decay.

b. Control Portfolio

1. Which Control Mechanisms to use?

There are informal and formal mechanisms which to platform owner can impose on the app developers to exert influence. There is not a predefined mix of control mechanisms which works. Moreover, the correct mix of control mechanisms depends on the context, on the app developers and what they are willing to accept as authority and the micro-architecture of the apps in the ecosystem. However, there are certain rules of the thumb which can help the platform owner create a control portfolio which serves the ecosystem's best interests. According to these rules, a platform's control portfolio should be simple, which implies minimized costs for both app developers and platform owner. It should be transparent and leave no room for ambiguity (e.g. explicitly defined criteria for accepted applications). It should be fair, realistic and comply with the values of the ecosystem. The platform owner should not forget that the architecture serves also as a powerful control mechanism, as long as the app developers comply to the API specifications.

The following table presents the various choices the platform owner has when designing a control portfolio. This can help the user pick control mechanisms using the rules described above. Any of the mechanisms shown in the table below can be choses as long as it fulfills three simple criteria:

1. It is needed.
2. If it can be substituted, the platform owner should pick the substitute instead.
3. It must fulfill the conditions described in the "Viable if" column.

Control Mechanism	Needed ?	Substitute ?	Viable if..
Gatekeeping	When creating performance metrics or monitoring processes is not possible	None; but prescreening app developers helps	• App developers accept a platform owner's authority to play gatekeeper • Compliance criteria are known and considered fair by app developers • Platform owner can costeffectively verify compliance
Process Control	Not needed if performance metrics are used	• Gatekeeping • Use of metrics based control • Allocation of app implementation decision rights to app developers	• Platform owner has credible expertise to dictate methods • Platform owner can verify process compliance by app developers
Metrics	Not needed if app developers retain app strategic decision rights	• Use of process control • If market determines winners and losers among app developers	Metrics are objectively measurable
Relational Control	• Fills gaps left by formal controls • Lower cost than formal controls	None; but prescreening app developers helps	• App developer churn is low • App developers and platform owners are bound by clan-like shared values

Table 2: Evaluating Individual Control Mechanisms for Inclusion and Exclusion. Adopted from [1].

Suggested Action:

Concerning the Control Portfolio, the Platform Owner is advised to act for the different Lifecycle stages as follows:

i. Lifecycle: Early Stages

- App developer compliance with the platform's interface is crucial.
- Communicate platform's vision and values to the app developers to promote Relational Control (2.3.2-a).
- Use Gatekeeping (2.3.2-b) only against harmful apps to the platform and complement with process control. The purpose of process control is to help app developers "pass" the Gatekeeping checks.
- Minimal screening on the app developers who join the platform.

ii. Lifecycle: Maturity Stage

- Screening of the app developers has the purpose to preserve the values of Relational Control.
- Communicate platform's vision and values to the app developers (Relational Control)
- Use Gatekeeping and complement with process control. The purpose of process control is to help app developers "pass" the Gatekeeping checks. The purpose of Gatekeeping is to ensure app developer compliance to the platform's interfaces.
- App developer compliance with the platform's interface is crucial.

iii. Lifecycle: Decline Stage

- The use of Gatekeeping and Process Control should be more intense at this point, due to increased complexity caused by the aging architecture.
- App developer compliance with the platform's interface is crucial.

c. Pricing Policies

The decisions given that a platform owner has to take are shown below. Symmetric pricing policies, access/usage fees and a moving pie-splitting scale are the alternatives over which the platform owner should decide. Asymmetric pricing policies also entail the decision of which side to subsidize. Also, there is one more factor concerning pricing which affects the ecosystem which is not enlisted below, because it is not a choice of the platform owner but rather a choice of the app developer. This concerns app licensing decisions, like multi-versioning of apps to generate revenue (e.g a premium version), and is mostly dependent on the app's micro-architecture and business model. However, the platform owner should bear in mind that choices in architecture might constrain viable app business models.

1. Symmetric or Asymmetric Pricing? If so, who to subsidize?

This decision is concerned with whether the platform owner will price both sides of the platform (app developers and end-users) or generate revenues from one side while the other side has negative or zero revenues (called the subsidized side).

Symmetric:
Pros: Revenues Generated from both sides.
Cons: More slow in generating network effects, especially if the platform is starting with both sides.

Asymmetric:
Pros: Acceleration of network effect creation.
Cons: One less revenue stream than symmetric, might include negative revenues for the subsidized side.

Suggested Action:
Concerning Pricing Policies, the Platform Owner is advised to take the following decisions for each Lifecycle Stage:

i. Lifecycle: Early Stages

At this Lifecycle stage, this decision depends on the platform owner. If the platform owner starts from the beginning with two sides, generating network effects is crucial. This justifies the use of asymmetric pricing. The only case where a platform owner can use symmetric pricing, is the case of the platform owner being an early dominant mover with a successful product. In all other cases, the platform owner should use asymmetric pricing.

ii. Lifecycle: Maturity and Decline Stages

Symmetric Pricing policies are possible at this point. However, asymmetric pricing policies might be desirable in horizontal envelopment target markets, where the platform owner should aim to increase the platform's adoption in the specific market niche. If the platform owner has started with subsidies at the earlier stages, the removal of these is advised to be gradual in the literature.

Subsidize App developers or End-users?

Subsidize App developers
Pros: Platform is more attractive to app developers
Cons: Platform is less attractive to end-users.

Subsidize end-users
Pros: Platform is more attractive to end-users.
Cons: Platform is less attractive to app developers.

Suggested Action
Concerning the subsidized side, the Platform Owner is advised to act as follows:

i. Lifecycle: Early Stages

At this point in the platform's Lifecycle, increasing the number of app developers who adopt the specific platform is critical. A big number of app developers means increased innovation rates and in extent a platform which has more possibilities to attract more end-users also. The platform owner should choose to subsidize app developers.

ii. Lifecycle: Maturity and Decline Stages

At this Lifecycle stage, a dominant design has emerged, so differentiation happens at the application level. Therefore, wide adoption of the platform by the app developers holds great value to the platform owner and subsidizing the app developers is advisable. However, the platform owner should always have a simple rule in mind when picking a side to subsidize: The subsidized side should be the one that is the most important for the other (e.g. a big number of end-users adds more value to the platform for the app developers instead of the value that is added by a big number of app developers for the end-users).

2. Use Access/Usage Fees?

This decision is concerned with whether the platform owner will charge end-users, app developers or both for providing access to the platform or for using it. The platform owner can either choose to charge either of the sides or to use free or even negative pricing (this means pay) to increase the platform's adoption. Usage fees will require from the platform owner to monitor usage using metrics, while access fees might take the form of a subscription/agreement.

Use Access/Usage Fees
Pros: Increased Revenues
Cons: Platform is less attractive.

Use Free or Negative Fees
Pros: Increased Attractiveness to either side
Cons: Decreased or even Negative Revenues.

Suggested Action:
Concerning Access fees, the Platform Owner is advised to take the following decisions for each Lifecycle Stage

i. Lifecycle: Early Stages

At this point, the platform wants to create Network effects as soon as possible. This makes app developers very valuable to the platform owner. The platform owner might also consider to boost the adoption of the platform by the end-users as it is currently in the early phases of the diffusion curve. Negative or Free usages fees are advised to increase the platform's attractiveness to both customer segments. To try and balance the high costs, the platform owner can introduce a third customer segment (e.g. advertisers) over which it can capitalize.

ii. Lifecycle: Maturity Stage

At this Lifecycle stage, the platform owner aims to increase product volumes and capture value. Therefore, access fees should be used if the platform is established. One example found in literature is an annual fee to the app developers, which acts as a token of commitment.

iii. Lifecycle: Decline Stage

At this Lifecycle Stage a platform is expected to be established, and access fees should continue to be used. Usually, platforms prefer charging the app developers over the end-users, for access to the platform or could charge them for usage. However, the literature warns against using both type of fees simultaneously.

3. Use Pie-Splitting Scale?

This decision is concerned with whether the platform owner will divide revenues with the app developers in a pre-agreed manner. If the platform owner chooses to do so, he should also choose between a stable or a moving scale. The following decisions are presented in the tables below.

Use Pie-Splitting Scale?		
	Yes	No
Pros:	Increased app developer attractiveness	Not specified in literature.
Cons:	Not specified in literature.	Decreased app developer attractiveness

Table 3: Pie Splitting Scale Decision Overview

4. Stable or Moving Scale?

Stable or Moving Scale?		
	Stable	Moving
Pros:	Not specified in literature.	Increased app developer attractiveness Promotes Innovation Rate
Cons:	Not specified in literature.	Not specified in literature.

Table 4:Stable or Moving Scale Decision Overview

Suggested Action:
Concerning the usage of a Pie-Splitting Scale and whether it is stable or moving, the Platform Owner is advised to take the following actions:

i. Lifecycle: Early Stages and Maturity Stage

The literature suggests the platform owner to use a Pie-Splitting Scale, as this serves as an indication to the app developers that the platform owner will not abuse his power which in turn increases the platform's App developer Synergy (2.2.3.2), and makes the platform more attractive. Furthermore, by using a Rising Scale (Revenue

split increases with good performance of an app in the market), the platform owner can increase the attractiveness of the platform to app developers. This move is suggested for the early stages of the platform's Lifecycle because it makes the platform more attractive to the app developers but it could also provide good results for a platform at the maturity stage. That is because a platform at the maturity stage would benefit from an increased rate of innovation (as differentiation mostly happens at the app level) and from a bigger number of app developers. A dominant platform however, would not need rising scales to attract app developers, because at this point it will have generated strong Network Effects in addition to a big market share.

ii. Lifecycle: Decline Stage

A stable pie-splitting scale is the usual practice at this Lifecycle stage. Rising scales are not useful at this point, as the keystone firm should prioritize in finding opportunities to enter a new market.

4.2.3 Structure

Goal:
The goals that the Platform Owner should strive to achieve concerning the ecosystem's Structure are presented below for each of the platform's Lifecycle Stages:

i. Lifecycle: All Stages

To provide support to the Architecture and Governance structures defined in the previous sections. The platform owner should create the necessary Organizational Capabilities within the firm to support the choices suggested in the sections above. Organizational Capabilities might include both organizational units and supporting technology tools (like IDEs, testing suites etc)

Decisions:

1. Provide Tools and Support to the app developers

This decision is concerned with whether the platform owner should invest in providing tools and support to the app developers.

Provide Tools and Support to the app developers?		
	Yes	No
Pros:	• Increased app developer attractiveness and adoption • Promotes Interoperability between platform and apps • Better Goal Convergence between platform owner and app developers	Not specified in literature.
Cons:	Not specified in literature.	Not specified in literature.

Table 5: Tools and Support Decision Overview

Suggested Action:

The Platform Owner is advised to act as follows concerning Tools and Support to the app developers:

i. Lifecycle: All Stages

Tools and support provided by the platform to the app developers is valuable at all Lifecycle stages to the app developers, so the platform owner is suggested to provide it. However, tools and support have an increased value for the ecosystem when the platform is at its early and maturity stages because it makes the platform more attractive to app developers, a big number of which is crucial in the early stages and highly valuable in the maturity stage.

2. Shift R&D focus to Real Options

This decision is concerned with whether the platform owner will have the R&D department of the firm to heavily focus on embedding Real Options (3.2.1).

Shift R&D focus to Real Options?		
	Yes	No
Pros:	• Enables Envelopment moves (3.2.3.3-a) • Enables Mutation moves (3.2.3.3-c) • Bolsters Plasticity (3.2.3.2)	Not specified in literature.
Cons:	Not specified in literature.	Not specified in literature.

Table 6: Real Options R&D Decision Overview

Suggested Action:
Concerning investment in Real-Options, the Platform Owner is advised to take the following decisions:

i. Lifecycle: Early Stages

Real Options have the most value in uncertain markets where volatility is high either due to converging new technologies or due to uncertainty about end-users' needs and expectations. That is why, the platform owner is suggested to heavily focus on embedding Real Options into the architecture at the early Lifecycle stages.

ii. Lifecycle: Maturity Stage

At this point in the platform's Lifecycle, market volatility is low and Real Options have decreased value in comparison with the early stages. However, the literature suggests that the platform owner should never completely abandon Real Options Thinking in any of the Lifecycle stages. At this Lifecycle stage, the platform owner should look for opportunities to exercise growth options (3.2.1).

iii. Lifecycle: Decline Stage

Since the market is a post-dominant phase, investing in Real-Options has very little value for the platform. However, in this Lifecycle stage, the platform owner should look for opportunities to use some of the Real Options already embedded in the design at the previous Lifecycle stages.

3. Evolve the platform's core?

This decision is concerned with whether the platform owner should create organizational capabilities charged with the task to identify and include generic and widely used (among the app developers) functionalities to the platform's core. Evolving the platform's core, also involves identifying commodity functionality and optimizing it for cost.

Evolve the platform's core?		
	Yes	No
Pros:	• Promotes App developer Stickiness (3.2.3.2) • Promotes Platform Synergy (3.2.3.2) • Promotes Durability (3.2.3.3-b) • Increases Platform's Attractiveness to App developers	Not specified in literature.
Cons:	Not specified in literature.	Not specified in literature.

Table 7: Platform's Core Evolution Decision Overview

Suggested Action:
Concerning Platform Core Evolution, the Platform Owner is advised to take the following decisions

i. Lifecycle: Early Stages

The platform owner is advised to evolve the platform's core at almost all stages of its Lifecycle. In the early stages, however, this activity might be even more intensive because of the high rate of Innovation happening at the Experimental Layer of the 3LPM (3.5.1.1).

ii. Lifecycle: Maturity Stage

Concerning core evolution, the platform owner should continue using the same strategy with the early stages with one addition: Organizational capabilities should be created to identify commoditizing functionality and transition it to the Commodity layer of the 3LPM. Functionality included in this layer is then suggested to be implemented using Open Source Software, in order to minimize owning costs.

iii. Lifecycle: Decline Stage

The keystone firm should allocate more resources in identifying commodity functionalities and optimizing them for ownership costs. Since, the Innovation rate in the whole market slowly draws to a halt, and innovation is imitated by rivals, the platform's differentiating layer will increasingly shrink. Therefore, differentiation at this Lifecycle stage happens within the platforms, concerning process and cost. The platform owner should switch commodity functionalities to Open Source Software.

4. Identify Target Markets for Horizontal Envelopment moves

This decision is concerned with whether the platform owner should create organizational capabilities within the keystone firm charged with the task monitor adjacent markets for horizontal envelopment (3.2.3.3-a) attack opportunities. A market/rival platform is considered adjacent when it shares a part of its user base with the platform of the keystone firm.

Create Organizational Capabilities for Horizontal Envelopment?		
	Yes	No
Pros	• Promotes Envelopment (3.2.3.3-a) • Increased expansion opportunities	Not Specified in Literature
Cons:	Not Specified in Literature	Not Specified in Literature

Table 8: Horizontal Envelopment Decision Overview

Suggested Action:
Concerning Horizontal Envelopment, the Platform Owner is advised to take the following decisions

i. Lifecycle: Early Stages

To the best of our knowledge, the literature does not address horizontal envelopment moves during the early Lifecycle stages. However, the lack of an established user base in both customer segments, makes horizontal envelopment moves seem counter-intuitive.

ii. Lifecycle: Maturity Stage

At this point in the platform's Lifecycle, the platform owner can engage in envelopment moves against adjacent platforms, as there is already an established user base which could be adjacent to other markets. Mature markets are ideal targets for such moves, and the platform owner is advised to create organizational capabilities within the keystone firm to identify such markets, and use a resource litmus test to evaluate if the platform can play the role of the disruptive technology in these markets. Furthermore, horizontal envelopment moves are a suggested way to bypass the network effects of rival platforms.

iii. Lifecycle: Decline Stage

The platform can engage in envelopment moves in this Lifecycle stage to sustain its market shares until it can enter the next S-curve in the market by embracing the

disruptive technology which by then will have been introduced. The literature emphasizes that this is nearly impossible for platforms which have adjusted their business models around incumbent technologies. Furthermore, it is considered as a lesser alternative in comparison with mutation moves (3.2.3.3-c).

5. Identify Target Markets for Mutation moves

This decision is concerned with whether the platform owner should create the organizational capabilities to identify markets that present ideal opportunities for a mutation move (3.2.3.3.-c). These markets are usually mature, in a post-dominant phase where the platforms have switched to process Innovation. Furthermore, the platform's own solution should pass the Resource Litmus Test (A1) for the target market.

Create Organizational Capabilities for Mutation?		
	Yes	No
Pros	• Promotes Mutation (3.2.3.3-c)	Not Specified in Literature
Cons:	Not Specified in Literature	Not Specified in Literature

Table 9: Mutation Support Decision Overview

Suggested Action:
Concerning Mutation, the Platform Owner is advised to take the following decisions

i. Lifecycle: Early Stages

Spending resources to identify mutation targets does not agree with the platform's strategy and is advised against in these Lifecycle stages.

ii. Lifecycle: Maturity Stage

Looking for mutation opportunities seems counter-intuitive for this Lifecycle Stage also. However, it would make sense if the platform was fighting a losing war, and there was a need to start over again in a new, fresh market.

iii. Lifecycle: Decline Stage

This Lifecycle stage makes the more sense for mutation moves according to the literature. A platform entering its Decline phase is usually at the brink of a death spiral and it has two choices. The first choice is to buy some time until it can make the leap to the next S-curve (3.1.1.1) by engaging in process innovation, cutting costs and engaging in horizontal envelopment moves (3.2.3.3-a). The second choice is to escape the dying market or a losing battle with a disruptive technological solution by doing a mutation move. The platform owner is advised to create such organizational capabilities to identify suitable target markets for the mutated platform, as the platform enters the Decline phase. In order to do so, the platform owner should be able to identify each of the Lifecycle stages, the transitions between them and also accept their consequences.

6. Maintain the platform's Interface

This decision is concerned with whether the platform owner should have organizational units charged with the task to slim down the platform's interface. The platform owner should pay attention not to break backwards compatibility with apps that evolve in a slower rate.

Slim down the platform's Interface?		
	Yes	No
Pros	• Slower rate of architectural decay	Not specified in literature
Cons	Not specified in literature	• Bloated platform interface • Increased Complexity

Table 10: Interface Maintainance Decision Overview

Suggested Action:
Concerning interface stability, the Platform Owner is advised to take the following decisions

i. Lifecycle: Early Stages

At this point in the platform's Lifecycle, there is no great need for such actions, as the platform is mostly expected to add new API's instead of having to remove unused legacy APIs.

ii. Lifecycle: Maturity and Decline Stages

The platform is expected at these Lifecycle stages to have legacy API's. The platform owner is advised to create organizational capabilities to identify such APIs and remove them from the platform to the extent possible without breaking backwards compatibility.

4.3 Overview

	Software Ecosystem Views			
	Strategy	Governance	Architecture	Structure

Early Decision Support Model	• Create Strong Network Effects
	• Attract App developers
	• Expand by Envelopment if possible
	• Strategic Incompatibility:
	  ○ Incompatible with rivals
	• Increase innovation rate
		• Decision Rights Partitioning:
		  ○ Centralized Platform Implementation Decision Rights
		  ○ Centralized Platform Strategic Decision Rights with app developer input
		  ○ Decentralized App Strategic Decision Rights
		  ○ Decentralized App Implementation Decision Rights with platform owner input
		• Control Portfolio:
		  ○ Use Relational Control and Gatekeeping supplemented with Process Control
		• Pricing Policies:
		  ○ If not dominant or starting with both sides: asymmetric pricing, subsidy side: app developers If dominant early mover with successful product: Symmetric pricing policies
		• Pie Splitting Scale – Rising
			• Modular Architecture- Decouple platform and apps
			• Internal Modularity:
			  ○ Adopt the 3LPM:
			  ▪ Decouple Experimental Layer
			  ▪ Decouple Differentiating Layer
			  ▪ Decouple Commodity Layer
			  ○ Decouple scalable functionality
			• Use Industry Standards for External Services
			• Use stable over dynamic APIs
			• Add new APIs for new functionalities
			• Minimal footprint rich in Real Options
				• Provide Tools and Support to app developers
				• Shift R&D focus on Real Options
				• Evolve the platform's core

Table 11:Early Decision Support Model (EDSM) Overview

	Software Ecosystem Views			
	Strategy	Governance	Architecture	Structure

Maturity Decision Support Model	• Scale upwards and horizontally by Envelopment
	• Increase Product Volumes (economies of scale)
	• Attract app developers
	• Strategic Incompatibility
	  ○ If dominant: Incompatible with rival platforms
	  ○ Else: One way compatible with the dominant platform
		• Decision Rights Partitioning:
		  ○ Centralized Platform Implementation Decision Rights
		  ○ Centralized Platform Strategic Decision Rights with app developer input
		  ○ Decentralized App Strategic Decision Rights
		  ○ Decentralized App Implementation Decision Rights with platform owner input
		• Control Portfolio:
		  ○ Use Relational Control and Gatekeeping supplemented with Process Control
		• Pricing Policies:
		  ○ Symmetric Pricing Policies unless BM indicates otherwise
		• Pie Splitting Scale – Rising or Stable
			• Modular Architecture- Decouple platform and apps
			• Internal Modularity:
			  ○ Adopt the 3LPM:
			  ▪ Decouple Experimental Layer
			  ▪ Decouple Differentiating Layer
			  ▪ Decouple Commodity Layer
			  ○ Decouple scalable functionality
			• Use Industry Standards for External Services
			• Use stable over dynamic APIs
			• Add new APIs for new functionalities
			• Switch commodity functionality to OSS
				• Provide Tools and Support to app developers
				• Evolve the platform's core
				• Identify target markets for envelopment moves
				• Maintain the platform's Interface

Table 12:Maturity Decision Support Model (MDSM) Overview

	Software Ecosystem Views			
	Strategy	Governance	Architecture	Structure

Decline Decision Support Model	• Retain user base
	• Mutate to new markets
	• Strategic Incompatibility:
	  ○ Two-way compatible
		• Decision Rights Partitioning:
		  ○ Centralized Platform Implementation Decision Rights
		  ○ Centralized Platform Strategic Decision Rights with app developer input
		  ○ Decentralized App Strategic Decision Rights
		  ○ Decentralized App Implementation Decision Rights with platform owner input
		• Control Portfolio:
		  ○ Use Relational Control and Gatekeeping supplemented with Process Control
		• Pricing Policies:
		  ○ Symmetric Pricing policies
		• Pie Splitting Scale – Stable
			• Modular Architecture- Decouple platform and apps
			• Internal Modularity:
			  ○ Adopt the 3LPM:
			  ▪ Decouple Experimental Layer
			  ▪ Decouple Differentiating Layer
			  ▪ Decouple Commodity Layer
			  ○ Decouple scalable functionality
			• Use Industry Standards for External Services
			• Use stable over dynamic APIs
			• Add new APIs for new functionalities
			• Switch commodity functionality to OSS
				• Provide Tools and Support to app developers
				• Evolve the platform's core
				• Identify target markets for mutation moves
				• Identify target markets for envelopment moves
				• Maintain the platform's Interface

Table 13: Decline Decision Support Model (DDSM) Overview

4.4DSM and Lifecycle.

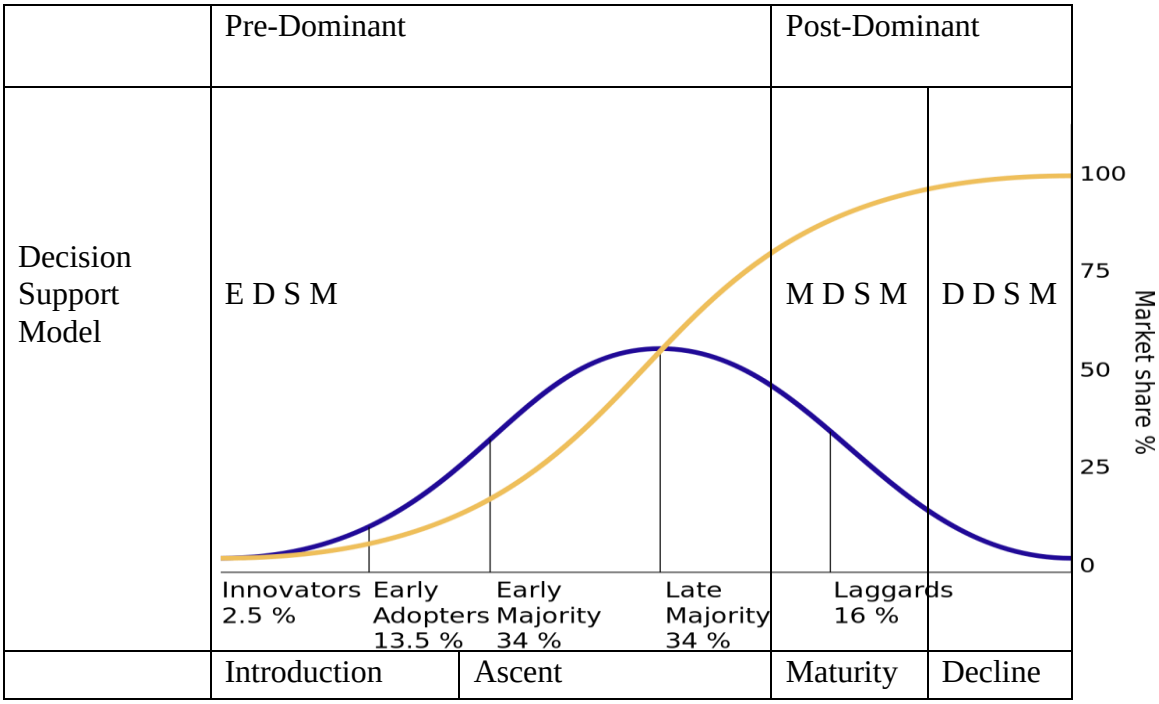

Figure 3: DSM and Lifecycle

5 Conclusions and Future Work

In this chapter, first, the measures taken to promote the reliability and validity of this study will be presented, followed by a subsection which describes ethical considerations of this study. Finally, the study's conclusion is given along with recommendations on further research.

5.1. Reliability and Validity

A research study's effectiveness is directly analogous to the rigor with which it is conducted [45]. Thus, to achieve academic rigor and present a trustworthy and authentic study, we are interested in enhancing the reliability and validity of it as these concepts are the key factors concerning the trustworthiness of any study [45]. The various strategies that are followed to ensure this study fulfills the standards of academic rigor are presented in the following sub-sections.

a. Reliability

The concept of a study's reliability is concerned with whether this study's results can be replicated by another researcher [45]. In qualitative studies, Reliability, as described above might be slightly problematic because there are differences between individuals. Furthermore, the same individual might be different in various times and, thus, give different answers. According to [45], "there is no benchmark by which to take repeated measures and establish reliability in the traditional sense". Driven by this, we focus our attention on maximizing the consistency between our result and the information collected during the literature study. For this purpose, the author reflects on the philosophical viewpoint that was used as theoretical lenses during the literature study.

b. Validity

There are several different views on Validity, which is concerned with whether the conclusions drawn from the data are valid. The most important of these views are Construct Validity, Internal Validity and External Validity.

Construct Validity, is concerned with whether the researcher leaves room for different interpretations of theoretical constructs found in literature, by the reader of the study. To avoid such threats to Validity, we use the most well-known and widely-used terms in this field of study, and provide sufficient explanation on the terms we use.

Internal Validity is concerned with the degree of the accuracy by which the results depict reality. To promote the internal validity of this study, we use the strategy of adequate engagement in information collection. An indication of the point when sufficient information has been collected is saturation. That is when new papers read during the literature study do not offer anything new in terms of information, and the same patterns are repeated. This strategy should be

complemented by purposefully looking for variations in the interpretation of the phenomenon under study [45].

To further promote credibility, we help the reader understand how we reached our conclusions by clarifying our philosophical worldview and approach to this problem. In this way, by informing the reader of the way that we rationalize, we help him understand how we extracted our findings from the literature study [45].

External Validity is concerned with whether it is safe to make generalizations based on the results of the study at hand and, in extent, how safe it is to apply these results in similar situations. Generalizability in its original statistical sense does not apply in qualitative studies. However, "Transferability" as explained in [45] is a better placed term. Transferability is concerned with the extend of applicability of a research study in similar cases. However, the reader is responsible in judging how well the results of a study at hand transfer to other cases. Thus, to promote External Validity, the author of the original study should give as much detailed information as possible about the context of the research. In this way, the readers of the study can make more accurate judgements in whether the findings of the original study are applicable to other cases [45]. Following this lead, we provide a rich description of the information collected in the Background section of this paper.

5.2. Ethical Considerations

During the whole phase of this research we follow an ethical code of conduct. The general principles and values that acted as ethical guidelines are:

- Honesty, in communicating results, methods and procedures etc.
- Objectivity, wherever it is required
- Integrity as in consistency of thought and action
- Professionalism in cooperation with companies and other personnel.

Confidentiality and informed consent wherever personal information is involved.

5.3. Conclusions and further research

Software Architecture plays a pivotal role in Software Ecosystems of all domains. It enables governance mechanisms to achieve integration and coordination between the platform owner and the third-party developers. It is also affected by and affects the organizational structure of the keystone firm. Furthermore, the decisions relevant to these views of a platform are dictated by the platform's strategy which is directly connected to the various phases of its Lifecycle.

The conceptual model, derived by the literature study shows exactly these relations and supports the decision-making process in designing and managing a platform. The model is compiled by studies of real world cases, and is created with the intent to depict a realistic decision making process in an industrial setting.

Even though we set out to study Industrial Cyber-Physical Ecosystems, we discovered two things: First, the literature does not cover Cyber-Physical Ecosystems at all, to the best of our knowledge, and second that these rules and decisions apply to all kinds of Software Ecosystems, as the same rules apply

concerning Software Ecosystems (Let us not forget that CPS are still directly relevant to Software), Business management and the market. Thus, a natural extension of this study would be to further expand the DSM with decisions specific to CPS, which might address unique problems and challenges which are inherent to CPS.

We believe that this study sets a foundation over which significant research could be based. Here we suggest some follow-up studies. The first suggestion we have for future work is to evaluate the Decision Support Model. An evaluation study, in a real word setting, with an Action Research approach is what we think that could produce the best results. The second suggestion is about finding out specific architectural tactics and patterns which would satisfy the requirements described in our conceptual model.

References

[1] A. Tiwana, Platform Ecosystems: Aligning Architecture, Governance and Strategy, Amsterdam: Morgan Kaufman, 2014.

[2] R. Baheti och H. Gill, ”Cyber-physical Systems,” The Impact of Control Technology, vol. 1, nr 1, pp. 161-166, 2011.

[3] J. Bosch, ”From Software Product Lines to Software Ecosystems,” In Proceedings of the 13th International Software Product Line Conference, Carnegie, 2009.

[4] E. Papatheocharous , J. Andersson och J. Axelsson, ""Ecosystems and Open Innovation for Embedded Systems: A Systematic Mapping Study.", "International Conference of Software Business", 2015.

[5] R. Taylor, N. Medvidovic och E. Dashofy, Software Architecture: Foundations, Theory, And Practice, Hoboken: John Wiley, 2010.

[6] S. Kim, D. K. Lu och S. Park, ”Quality-driven architecture development using architectural tactics,” Journal of Systems and Software, vol. 82, nr 8, pp. 1211-1231, 2009.

[7] B. Furht och A. Escalante, Handbook of Data-Intesive Computing, Springer Science and Business Media , 2011.

[8] Y. Demchenko, N. Canh och P. Membrey, ”Architecture framework and components for the big data ecosystem,” Journal of System and Network Engineering, pp. 1-31, 2013.

[9] C. A. Mattman , D. J. Crichton, N. Medvidovic och S. Hughes , ”A software architecture-based framework for highly distributed and data intensive scientific applications,” Proceedings of the 28th international conference on Software engineering, 2006.

[10] J. Wang, D. Crawl och I. Altintas, ”Kepler+ Hadoop: a general architecture facilitating data-intensive applications in scientific workflow systems.,” Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science, 2009.

[11] Z. Shu, D. Wan, D. Li och D. Zhang, ”Cloud-integrated Cyber-Physical systems for complex industrial applications,” Mobile Networks and Applications, vol. 21, nr 5, pp. 865-878, 2015.

[12] M. Engelsberger och T. Greiner, ”Software architecture for cyber-physical control systems with flexible application of the software-as-a-service and on-premises model.,” Industrial Technology (ICIT), 2015 IEEE International Conference, 2015.

[13] K. Dwivedi och K. D. Sanjay, ”Analytical review on Hadoop Distributed file system.,” Confluence The Next Generation Information Technology Summit (Confluence), 2014 5th International Conference-. IEEE, 2014.

[14] S. Ghemawat , H. Gobioff och S. T. Leung , ”The Google file system,” ACM SIGOPS Operating Systems Review, vol. 37, nr 5, p. 29, 2003.

[15] K. Shvachko, H. Kuang , S. Radia och R. Chansler, ”The Hadoop distributed file system.,” 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies, 2010.

[16] M. Zaharia , M. Chowdhury, M. J. Franklin , S. Shenker och I. Stoica , ”Spark: Cluster Computing with Working Sets,” HotCloud’10 Proceedings of the 2nd USENIX conference on Hot topics in Cloud Computing, 2010.

[17] Y. Zhang, M. Qiu, C. W. Tsai, M. M. Hassan och A. Alamri, ”Health-CPS: Healthcare Cyber-Physical system assisted by cloud and big data,,” IEEE Systems Journal, pp. 1-8, 2015.

[18] S. Karnouskos, ”Cyber-physical systems in the smartgrid.,” Industrial Informatics (INDIN), 2011 9th IEEE International Conference, 2011.

[19] H. Li, L. Lai och H. V. Poor, ”Multicast routing for decentralized control of cyber physical systems with an application in smart grid,” IEEE Journal on Selected Areas in Communications, vol. 30, nr 6, pp. 1097-1107, 2012.

[20] S. Sridhar, A. Hahn och M. Govindarasu, ”Cyber–physical system security for the electric power grid,” Proceedings of the IEEE, 2012.

[21] J. Zhao, F. Wen , Y. Xue, X. Li och Z. Dong, ”Cyber physical power systems: architecture, implementation techniques and challenges.,” Automation of Electric Power Systems, vol. 34, nr 16, pp. 1-6, 2010.

[22] L. Parolini, B. Sinopoli, B. H. Krogh och Z. Wang, ”A Cyber–Physical systems approach to data center modeling and control for energy efficiency,” Proceedings of the IEEE, 2012.

[23] A. A. Cardenas, S. Amin och S. Sastry, ”Secure control: Towards survivable cyber-physical systems,” Distributed Computing Systems Workshops, 2008. ICDCS'08. 28th International Conference on IEEE, 2008.

[24] J. Lee, B. Bagheri och H. A. Kao, ”A cyber-physical systems architecture for industry 4.0-based manufacturing systems,” Manufacturing Letters, vol. 3, pp. 18-23, 2015.

[25] Y. Tan, S. Goddard och L. C. Perez, ”A prototype architecture for cyber-physical systems,” ACM Sigbed Review, vol. 5, nr 1, p. 26, 2008.

[26] L. Wang, ”Machine availability monitoring and machining process planning towards cloud,” CIRP Journal of Manufacturing Science and Technology, vol. 6, nr 4, pp. 263-273, 2013.

[27] L. Wang, M. Törngren och M. Onori, ”Current status and advancement of cyber-physical systems in manufacturing,” Journal of Manufacturing Systems, vol. 37, pp. 517-527, 2015.

[28] J. West, W. Vanhaverbeke och H. W. Chesbrough , Open Innovation : Researching A New Paradigm, Oxford: OUP Oxford, 2006.

[29] M. Bogers , A. K. Zobel, A. Afuah, E. Almirall, S. Brunswicker , L. Dahlander, L. Frederiksen, A. Gawer , M. Gruber, S. Haefliger och J. Hagedoorn, ”The open innovation research landscape: Established perspectives and emerging themes across different levels of analysis,” Industry and Innovation, vol. 24, nr 1, pp. 8-40, 2016.

[30] N. J. Foss och T. Saebi, ”Business models for open innovation: Matching heterogeneous open innovation strategies with business model dimensions,,” European Management Journal, vol. 33, nr 3, pp. 201-213, 2015.

[31] H. W. Chesbrough och M. M. Appleyard, ”Open innovation and strategy,” California management review, vol. 50, nr 1, pp. 57-76, 2007.

[32] K. Manikas och K. M. Hansen, ”Software Ecosystems – A Systematic Literature Review,” J. Syst. Software, vol. 86, nr 5, pp. 1294-1306, 2013.

[33] IEEE, ”Systems and software engineering - Vocabulary,” 2010.

[34] J. Bosch och . P. Bosch-Sijtsema, ”From integration to composition: On the impact of software product lines, global development and ecosystems.,” Journal of Systems and Software, vol. 83, nr 1, pp. 67-76, 2010.

[35] A. Leonard, ”The viable system model and its application to complex organizations,” Systemic practice and action research,, vol. 22, nr 4, pp. 223-233, 2009.

[36] K. Rong, Y. Lin, Y. Shi och J. Yu, ”Linking business ecosystem Lifecycle with platform strategy: a triple view of technology, application and organisation,” International journal of technology management,, vol. 62, nr 1, pp. 75-94, 2013.

[37] H. H. Olsson och J. Bosch, ”Strategic Ecosystem Management: A multi-case study on challenges and strategies for different ecosystem types.,” Software Engineering and Advanced Applications (SEAA), 2015, 41st Euromicro Conference on IEEE, 2015.

[38] I. van den Berk, S. Jansen och L. Luinenburg, ”Software ecosystems: a software ecosystem strategy assessment model,” In Proceedings of the Fourth European Conference on Software Architecture: Companion Volume (pp. 127-134). ACM., 2010.

[39] J. Bosch, ”Architecture challenges for software ecosystems,” In Proceedings of the Fourth European Conference on Software Architecture: Companion Volume, 2010.

[40] S. Jansen, ”How quality attributes of software platform architectures influence software ecosystems,” In Proceedings of the 2013 International Workshop on Ecosystem Architectures , 2013.

[41] J. Bosch, ”Achieving Simplicity with the Three-Layer Product Model,” IEEE Computer, vol. 46, nr 11, pp. 34-39, 2013.

[42] M. Iansiti och R. Levien, Keystones and dominators: Framing the operational dynamics of business ecosystems, 2002.

[43] H. B. Christensen, K. M. Hansen, M. Kyng och K. Manikas , ”Analysis and design of software ecosystem architectures–Towards the 4S telemedicine ecosystem,” Information and Software Technology, vol. 56, nr 11, pp. 1476-1492., 2014.

[44] S. Jansen och M. A. Cusumano, Defining software ecosystems: a survey of software platforms and business network governance. Software ecosystems: analyzing and managing business networks in the software industry, 2013.

[45] J. Axelsson, E. Papatheocharous och J. Andersson , ”Characteristics of software ecosystems for Federated Embedded Systems: A case study.,” Information and Software Technology, vol. 56, nr 11, pp. 1457-1475, 2014.

[46] S. Durst och P. Poutanen, ”Success factors of innovation ecosystems–initial insights from a literature review.,” In Proceedings of Co-Create 2013: The Boundary-Crossing Conference on Co-Design in Innovation, Espoo, Finland, 2013.

[47] E. Raymond , ”The cathedral and the bazaar.,” Knowledge, Technology & Policy, vol. 12, nr 3, pp. 23-49, 1999.

[48] P. Runeson och M. Höst, ”Guidelines for conducting and reporting case study research in software engineering,” Springer, 2009.

[49] P. Anderson och M. L. Tushman, ”Technological discontinuities and dominant designs: A cyclical model of technological change,” Administrative science quarterly, pp. 604-633, 1990.

A Appendix 1

A.1. The Resource Litmus Test

Resource Property	Competitive Advantage	
	Creates	Sustains
Valuable?	•	
Rare?	•	
Inimitable?		•
Nonsubstituable		•

Table 14: The Resource Litmus Test. Adopted from [1]

Valuable: Is it of value in the platform's market and industry?
Rare: Do very few rival platforms have it?
Inimitable: Is it difficult (prohibitively costly or time-consuming) for a rival platform to imitate?
Nonsubstitutable : Can something else substitute for it?