Quest for Control: Managing Software Development in Networked Operating Environments

Kari Koskinen
Aalto University
kari.m.koskinen@aalto.fi

Sonja Hyrynsalmi
LUT University
sonja.hyrynsalmi@lut.fi

Matti Rossi
Aalto University
matti.rossi@aalto.fi

Kari Smolander
LUT University
kari.smolander@lut.fi

Abstract

Instead of developing software purely within the confines of one company, software companies increasingly procure many of the functionalities of their software from external entities and actors via system integrations and utilizing resources provided by external application programming interfaces (APIs). In addition to the benefits that can be reaped via integrations and working in cooperation with other companies, this type of networked software development leads to a reduction of control for the individual companies. As a result, companies need to resort to specific strategies and practices that reduce the risks emerging from lack of control. By utilizing data collected from Finnish software companies, we map the factors that cause reduction of control, study why companies give away control, and identify the challenges surfacing from it. To tackle these issues, we identify two strategies that software companies can take to counter the reduction of control.

1. Introduction

In modern software development, integrations to external systems as well as utilization of tools and resources provided by external actors and entities are often necessary. Instead of building everything in-house, certain functionalities as well as resources such as data can be obtained from external sources. In most cases, the development of software occurs on top of external development environments and utilizes widely adopted digital platforms, infrastructures, and entire ecosystems [1, 2]. These developments have resulted in the expansion of software development projects beyond the limits of a single firm into ecosystems consisting of various actors and technologies, and the developed software in many cases resembles more of a constellation of externally provided functionalities and other resources combined in a particular manner [3]. As a result, these factors have led to the creation of project and software development structures that can be viewed as networks of actors and resources that do not adhere to strict organizational or other boundaries. The technological nodes in these structures are the integrations to systems that offer tools, functionalities, data, and other resources for others to use, for example, via application programming interfaces (APIs) [4]. In addition, these resources themselves may rely on other external technologies, further emphasizing the ecosystem-like character of software development projects consisting of networks of different actors and resources.

An example of this can be seen in the utilization of digital platforms, on which software applications are built. The company developing the application functions as a complementor to the platform [5], and in relation to the platform owner, it can be seen as a non-focal actor that is highly dependent on the platform in regard to the development and functioning of the application [6]. In addition to the resources provided by the platform, the application may draw functionalities from other sources and use data from different external entities while having at least some parts of the software application in public cloud infrastructures. The software company becomes dependent on all those external actors and on the decisions that these entities make, yet it has little or no direct control over the resources or the decisions [7].

In this paper, we illustrate that as the technological and organizational boundaries become lower [8] and available technological resources and project partnerships increase, situations in which software companies have less control over the software they are developing are more frequent. This reduction of control is driven by factors occurring on two fronts. The first one is largely technological and occurs as a result of relying on externally provided technological resources such as digital data and functionalities. The second evolves from the manner in which software projects are organized, as those projects may consist of several actors and entities instead of taking place solely within the premises of one company.

URI: https://hdl.handle.net/10125/71342
978-0-9981331-4-0

As these companies have less direct control over the software that is being developed, the argument put forward here is that this may lead to loss of stability and predictability in software development. Stability is lost as there are a number of external and, hence, to some extent, non-controllable factors, and a change in those factors can require non-planned changes to the software. Predictability is lost because it can be difficult to foresee what kinds of changes will occur in those factors in the future. This inability to predict the changes can be further exacerbated as these external factors may on their part similarly depend on another set of actors and resources external to themselves.

The research shares some common characteristics with the literature on platforms [4, 9]; yet, instead of viewing this from the perspective of the resource owners and bigger actors, it focuses on the companies using the resources and looks for ways that these companies can mitigate the challenges emerging from this loss of control. The research questions set for the paper are the following: first, why do software companies engage in practices that lead to reduction of their control over the developed software? Second, what are the benefits and challenges following this reduction? Third, how do these companies mitigate the challenges brought upon them by the reduction of control? In our research, we aim to answer these questions by focusing on Finnish software companies that resort to integrations with external systems and utilize external resources in developing software. As noted, these companies can be largely seen as resource takers, similar to the non-focal actors of major platforms, and resource providers as described by Selander et al. [6]. They are largely unable to have direct control over the utilized resources or resource owners because of a lack of power to do so. At the same time, they need to operate in an environment where integrations to external actors and technologies are in many cases essential [10, 11]. However, simultaneously, these companies value stability, which might be in short supply due to the organizational and technological reduction of control.

2. Trajectory of Networked Software Development

The type of networked software development discussed above refers to a development environment in which individual companies rely on other companies to develop software, for example, in the form of partnerships and subcontractors. In addition, the developed software utilizes heavily external resources and functionalities. Traditionally, networked software development has either been used to refer to software development projects that have resulted from outsourcing or otherwise moving software development to different locations [12, 13], or software development that takes place in open source communities that consist of various heterogeneous actors possessing different roles [14]. Both are examples of how the development of software is done in a networked manner. In addition to these, increasingly, the software artifacts themselves are becoming networked as they are built on external platforms [15], utilize cloud [16], or otherwise rely on externally provided data and functionalities provided, for example, via APIs [17].

Behind these developments are the increasing digitization of information and socio-technical processes, as well as the need to develop information systems faster. As defined by Yoo et al. [18], digitization can be understood as encoding information into a digital format, which among other things enables processing such information via pre-programmed instructions. As this kind of information is quite agnostic in terms of the devices and systems in which it is used and can be altered in various ways, it can also be shared and moved from one system to another over information networks with relative ease. This has led to further digitalization, in which socio-technical structures are increasingly mediated by digital artifacts or relationships [18]. Due to the increasing appearance of APIs, as well as connecting both physical and digital resources to the network, communication has also begun to take place between artifacts in addition to people using the products and services [8]. APIs offer data, functionalities, and technological resources for developers to use [4, 17]. Via increasing amounts of external and internal system integrations combined with the overall provision of APIs, different actors and entities such as digital platforms can provide other developers and software-based products and services resources that perform key functions in those systems [15].

Linked to this, modularity and the move toward modularization of both software and organizational processes have facilitated the sharing of tasks and functions across organizations by splitting those into specific units or compartments. The architecture of a software shows the product's fundamental structure, utilized components, and the interfaces between those, which together form the product's functionalities [19]. The product can be divided into modular components, each of which has a particular functionality and is responsible for a part of the functioning of the product [19–21]. Modularity of software refers to the degree to which the components of that product can be separated and combined in different ways [21].

Modularization is decided based on factors such as distribution of design work, available technology, manufacturability, and maintainability [20–23]. It has

thus partly led to reorganization of work by dividing it into different areas and tasks, which is impacted by factors such as how companies operate or are structured, for example, in terms of production [24]. The fact that each function is placed into its own unit also enables the development of those units externally and therefore facilitates processes such as outsourcing [25]. Modularity contributes to vertical deintegration of a firm [26], since, as in the logic of outsourcing, certain aspects or areas can be left for external actors [27]. One example of this is the development of additional services and components by third parties [11], which has enabled the creation of product or service ecosystems. In these ecosystems, the applications developed by third parties are seen as complementing the platform, which provides the applications technological resources that these rely on in their functioning [28].

In our view, the concept of modularity is at the core of software development and enables the development and management of large-scale software projects by a variety of actors [20, 29]. External and internal system integrations combined with the overall provision of APIs, different actors, and entities such as digital platforms can provide other developers and software-based products and services resources that perform key functions [15]. These resources themselves may derive part of their functionality from other similar resources, thus creating a development environment that is highly interlinked through various direct and indirect technological connections. In this network, certain actors, such as major platform companies, function more as resource-givers and smaller software companies as resource takers by relying on the provided resources to develop their own software. From the perspective of a software company that occupies a peripheral position in relation to the resource providers in the sense that it uses those resources but does not provide them, the operating environment and the software development projects become more fragmented, consisting of various actors and resources.

To a fair extent, this network of different actors, digital functionalities, and data enables processes and projects to become less bounded and more interconnected, similar to the infrastructural factors that support those innovations [30]. By being connected and digital, these systems and various digital products are subject to being continuously edited and changed, which also offers avenues for other actors to join, for example, by expanding the systems and products in terms of their existing functionalities or transferring those systems and products to new contexts and environments [31].

As the operating environment resembles more of a network than a hierarchical structure, the question that emerges is who has control over the systems or how to control them. In response to this, technological modularity has been seen as guarding against lack of control as the inputs and outputs of the modules are relatively standardized and clear [32] and requiring no intervention from a particular actor. However, the question of control and overall governance has remained a central topic particularly in relation to platforms [33, 34].

Especially in highly networked software development ecosystems that consist of various actors and technological resources, having control in some form or another over the environment also functions as a source of stability and predictability over the developed software. The requirement for stability as well as predictability is therefore central for the actors using the resources. At the same time, by utilizing external resources, control is being lost, as these actors do not have any say in how those external resources are developed or maintained. Research so far has focused more on the platform owners, highlighting issues related to, for example, platform governance or cultivating exponential growth by the platform owner [3, 35]. With some exceptions, there have been fewer studies looking at non-focal actors who rely on these platforms to make sure their own applications continue functioning. In addition to looking at why companies engage in activities that result in reduction of control, this paper seeks to provide further insights into how the resource-taking companies operate in networked software development environments and how these companies can bring predictability and stability to the networked environment while still reaping the benefits from it.

3. Methodology

In order to answer the set research questions, we adopted a qualitative approach and interviewed people working in Finnish software companies. A total of 20 interviews were conducted. The interviewees consisted of developers and managers as we aimed to cover both the technological and organizational dimensions of networked software development. In addition to the companies being Finnish and working in areas closely linked to software development, all of the interviewees had experience in software integrations, and the companies they represented were, in most cases, resource takers and had little say about how the utilized resources were to be maintained or developed.

The interviews were semi-structured and lasted from 60 to 90 minutes. The interviews were recorded, transcribed, and coded using Atlas.ti. The data were analyzed by using thematic analysis, as coding resulted in codes that could be further linked to subthemes of networked software development overall as well as strategies meant to counter the identified reduction of control. The analysis was guided by the interview

questions as well as the research questions. In total, 211 codes were generated, which functioned as a basis for subthemes, such as "best practices," "challenges," "change," benefits," and "differences in integrations." The subthemes provided the foundation for the findings that enabled us to answer the set of research questions.

Overall, the research took the abductive approach in investigating the topic. In other words, there was no intention to test existing propositions nor to generate theory directly from data alone, but more to analyze the data and develop the research by concurrently visiting theory and empirical observation, and instead of generating new theory, we aimed at taking existing frameworks and developing them further in relation to our own research [36]. From the codes and themes, we identified the emergence of the phenomenon of networked operating environments in software development. After this, we aimed to see how this environment could be better understood theoretically and to identify from the literature how this environment has evolved and the issues involved with it, such as the notion of control and its paradoxical relationship with enabling generativity and flexibility.

4. Findings

The analysis of the data focused first on identifying factors that have led to reduction of control for individual software companies. This emerged from interviewees' citing situations and events in which their company had resorted to technologies and actors that were external to the company itself. We then moved to study the benefits that were obtained from relying on external technologies and actors. The next step was to analyze the specific challenges that the reduction of control caused, which was followed by looking at strategies and practices that allowed the companies to compensate for the reduction of control and mitigate the possible risks emerging from it.

The reduction of control for individual software companies took place along two dimensions, namely, in relation to technological reasons such as utilized external technological resources and organizational factors such as partnerships and use of subcontractors. The reduction of control due to technological factors to some extent facilitated the creation of networked organizational environments. For example, product modularization and the possibility for external integrations also made cooperation among companies in software development more feasible. As a result, our research was able to distinguish between these two dimensions, and we found that the overall reduction of control for individual software companies can emerge from both technological and organizational factors.

4.1. Factors Contributing to Reduction of Control

Our analysis of the data revealed that there were several factors that led to a software company being unable to fully control the software it was developing. One clear example of this occurring was when companies developed applications for a particular operating system and hence relied on functions and data provided by the platform. A similar type of reduction of control occurred in relation to utilizing public cloud companies such as Amazon Web Services or Microsoft Azure. The use of these resources enabled many software companies to avoid directly owning hardware such as servers, while also obtaining the added benefit of having a range of functionalities such as analytics tools or machine learning capabilities at their disposal.

"We have been thinking about moving those to AWS [Amazon Web Services], because they probably also have better tools for documenting, and at the same time, we could have that separate from the customers' systems" (interviewee 6 (int6)).

In addition to these, another contributor toward reduction of control emerged in the form of data and functionalities that were integral in making the software function as intended. Examples of the functionalities could be seen, for example, in utilizing maps or authentication services in the developed software artifacts, or regarding data, receiving it, for instance, from institutions such as transportation operators providing data about schedules or movements of their fleet.

"Thinking about our software, the first thing that comes to my mind [in terms of externally acquired functionalities] are the location and map-based services that we use, as they play a big role in our products" (int7).

As noted, the common factor for all of these was the requirement for integration into systems and sources that resided outside the software company and tapping into those sources. This has led to the establishment of technologically mediated connections to the entities providing those technological resources and services, and overall the utilization of external resources.

To a certain extent, the reliance on external technological resources provided the groundwork for also utilizing external partners and actors on an organizational level. As the systems connected various actors, external actors also had to be involved and were part of the software development projects. Some external actors also acted as middlemen toward other actors.

"If you think of a normal project, there are quite a few actors already involved via our customers' own networks, and all of those need to be taken into account

when we are building the new system, and we need to deal with those third parties as well even though they might not be directly related to our project" (int12).

Reduction of control due to organizational factors was also witnessed in the partnerships between software companies as they collaborated in the development of software. Companies also formed partnerships, for example, in competing or applying for funding for software projects and developing those in groups. The number of participants in these partnerships differed considerably, varying from one to several dozens.

"In this one project, we had something like forty plus IT companies involved" (int4).

Similarly, sometimes the customer for a developed software was an alliance of different entities and consisted of several companies, each of which occasionally had interests that were not always aligned with those of the others. Although the latter factor did not necessarily lead directly to reduction of control for the company developing the software, it had consequences in terms of having to serve various and sometimes differing interests, possibly also complicating the further development of the software.

What also contributed to this was the requirement to serve multiple stakeholders from within one system.

"There were quite a few different [actors involved], for example, the telecom operators. Then we had to take into account the public institutions, then via the companies their different units such as factories, which also had their own IT systems" (int10).

In addition to partnerships, a more traditional form of subcontracting other companies or customers using various companies to carry out areas of the development of the software could also cause difficulties in managing the whole development process.

"The worst thing is when it turns into something like a developer, who tries to use the interface, sends a mail to the customer saying that the interface does not work, the customer forwards it to the other software company, which says something completely different to the customer. Then that response comes to us, and it becomes this game of ping-pong where the customer is at the middle. So, having some visibility would definitely be useful" (int13).

Overall, the key technological and organizational factors contributing to the reduction of control formed something of a continuum, in which the more the companies resorted to the abovementioned factors, the more the control of the development of the software and its functioning was handed over to forces and actors residing outside the confines of the company itself.

4.2. Benefits and Challenges Resulting from Factors Causing Reduction of Control

4.2.1. Benefits. By resorting to external resources and actors in software development, the software companies obtained multiple benefits. A clear example of this was the ability to use resources such as maps that might have otherwise required significant investments or would simply be out of reach for many of the companies.

"There are fewer cases nowadays where you simply cannot do something, or it does not work. Back in the days, there were quite often those that the technology was not quite ready or something else, and in this world of integrations there really aren't those show stoppers" (int10).

Integrations between different systems also allowed automatization of processes.

"I think the biggest value is in being able to automatize work along the whole process chain […] For example, because of the connected systems, there is no need for an electrician to go and switch on electricity. It can all be done remotely" (int12).

The ability to save costs and to respond to fluctuating demand were noted as being among the benefits of the utilization of public cloud companies. Instead of having to invest in hardware and manage that in-house, these cloud companies offered a feasible way for the software companies to have the required computational resources at their disposal, also to be able to scale up when needed. Additionally, the public cloud providers offered additional functionalities for the software companies and enabled faster development cycles overall.

"My opinion is that they provide a nice platform on top of which to build applications and solutions really fast and in a very convenient manner, which can be seen also in costs […] What you do need to take into account are the interests linked to having control, and that if in the wrong hands damage can be done" (int12).

In terms of organizational factors, the benefits obtained from partnerships and the like shared some characteristics with the benefits obtained from technological factors. For example, subcontractors enabled companies to direct their own resources to areas where their main expertise resided and, in some instances, also to meet the set deadlines for the development of the software. Partnering with others also allowed companies to expand their own offerings to areas where they did not have much experience.

"It is quite typical that we do things to a point we can, and then partner with someone who is particularly good in the technologies that have been picked for the project" (int10).

Forming partnerships also enabled smaller companies to compete for and gain projects they alone would not have been able to do.

"It was good for the project that each participating company provided the people who were the best for that particular job" (int13).

Finally, another benefit of partnering was that if problems emerged, they were in some sense shared, as the development of the software artifact was dependent on the correct functioning of all its constituting parts.

4.2.2. Challenges. By relinquishing control, challenges followed. In terms of technological challenges, since resources were derived from external sources, the software companies were unable to directly dictate or even impact the decisions concerning the development and evolution of those technological resources. If a decision was made about changing a resource in some way by the entity hosting the resource, the companies utilizing the resource often had little choice other than to accept the changes as they were and update their own software accordingly.

"Well, you have to live according to their [software development kit (SDK) provider] updates, and test your system when they update, just recently when there was an update some of our functionalities stopped working, or then when certain functionalities are deprecated and that requires work from our end" (int5).

Because companies had little say over how the externally provided resource would evolve or function, this led the companies to tweak or fork the resource in a manner that was not entirely intended by the host of the resource. In cases where too much forking occurred, it was possible that, as the obtained resource was updated by the host organization, the software utilizing the fork encountered errors and was unable to function as intended. Similarly, sometimes, the resources were difficult to combine with other resources.

"We would like to move the mobile solution to React Native, which can be used on both iOS and Android, but it is difficult because we have the other SDK in this" (int5).

In addition, as the software was linked to other systems and tools, the problems spread more easily and impacted all the integrated systems and software, and the software and its developers were largely dependent on other actors to fix the problems.

"Every time you work in this kind of environment where the system should always work or the entire facility comes to a halt, and as you have integrations to other systems that are critical for the functioning of the system, those might mean that if you don't get the data from there, there is nothing the facility can operate on" (int17).

Integrations into external resources came with the added risk of making the software more vulnerable to external malfunctions. When errors occurred, receiving support from the host entity was occasionally seen as challenging, leaving the companies unable to fix the problem. This caused delays in the development of the software or required the companies to build additional software components to prepare for the errors.

"It is often a challenge that we state that we need this type of feature to make this work, and even though we have the same customer, the other company just does not have the resources, and they cannot give you the support for building that feature until only in some months' time" (int14).

Another challenge that resulted from the utilization of external resources and reductions of control over the software development was the ability to test the software and its external parts, which in some cases was completely lacking.

"Sometimes there is no testing environment or it is not updated, or it's down for several days, and in terms of integrations, they need to be tested, and fixing issues can take quite some time, and you cannot just change your system so that the integrations stop working" (int14).

Overall, if the reliance on external resources was too great, that also meant that those resources were very difficult to manage. Being able to communicate effectively and be aware of the changes done for each of the resources was not always easy. Also, the more partners and external resources there were, it became more cumbersome to capture the big picture of the software's development.

"Another challenge is working with several actors […], you need to have the overall picture clear on what it is that you are actually trying to develop" (int12).

Software development projects conducted in cooperation with partners also led to increased dependency among them, which also meant that problems of one company became, in this way, shared by others.

"Sometimes it gets quite strange. For instance, there was this one problem we were trying to solve with a customer for months, and then it turned out that the data that came from the customer's customer was done in a manner that did not follow the standards very strictly, and since it worked with some programs but not with others, it turned out that the programs in which it did work were not too picky about the format the data came in" (int11).

Overall, most of the problems were seen as a result in difficulties in communication.

"It happens every time in projects with third parties, or when we have to integrate into another system that requires some changes. The communication

just does not usually work […] It can be something like it just takes time to get replies or support" (int14).

In addition, the collaboration and cooperation between partners required certain common tools and frameworks which the partners that had no prior experience had to learn and adopt first.

"I have not had difficulties learning those, but if people in companies are not yet using those, then you have to first teach those how to use software like Jira in order to have a common view of the project in one place" (int4).

4.3. Strategies and Practices to Counter the Challenges

To counter the challenges resulting from reduction of control, the software companies resorted to different strategies and practices both on the technological and organizational levels. One was to simply try to build as much in-house as possible.

"Occasionally there have been cases where we have decided to build something ourselves, even if there was already something available, though that has been often because we have not been able to integrate that functionality very well, and even when building ourselves, we look if there are some components that could be obtained elsewhere" (int17).

What is noteworthy is that some of the same factors, which led to the reduction of control, also contained mechanisms that helped to counter the challenges and lessen the negative impacts from the lack of control. One example of this was the major public cloud companies, which were often seen as generally trustworthy and stable because of their size and resources, but also because of the competition among them. All the major cloud companies were viewed as being able to provide a large set of functionalities and services and being relatively easy to use with reasonable levels of support available.

There was an indication that it was better to utilize resources that had alternatives available if something went wrong with the use of the resource. However, switching from one resource to another was often seen as requiring a significant amount of work and adaptations to the other areas of the software under development.

"We quickly realized that it was necessary to build connections to at least two different operators, since if there was a failure in one at least the other one worked okay" (int19).

If one were to choose between sources providing similar resources or functionalities, such as maps or authentication services, actors seen as well-established provided a somewhat safer option in terms of continuity and support availability. Although not directly stated, it could also be argued that relying on functionalities, which one's key competitors also utilized, meant that, if there was a problem with a particular functionality, the competitors were likely to face the consequences as well.

Overall, open source solutions were occasionally seen as less risky than proprietary ones, especially if the continuity of the host organization was of concern to the company utilizing the resource. Naturally, this ability of an open source to provide stability depended on the type of resource it provided; however, having access to the source code gave the companies time if unforeseen disruptions occurred or the resource was no longer actively maintained.

"Well, if it is open source, there could be the thing that then it is easier to fix, like if there is something in the SDK that the provider does not fix, you can do it yourself" (int5).

However, also in the case of an open source, it was necessary to evaluate other aspects of the open source project, such as how active the community was running the open source project.

Many of the interviewees expressed the importance of standards and common procedures as those have offered clarity and made the cooperation between different companies and integrations into different systems easier. Standards have established the norms and rules for how software and related components are to be built and developed, and have further enabled more efficient communication between partners.

"Just that there is the standard, so that you can just watch and see that this is how the process goes, without having to study it [the standard] first for hours […], and overall, if something needs to be done, is to provide standards which are globally shared and became de facto, that is, something that needs to be supported" (int15).

Linked to this, developers especially cautioned against tweaking or excessively forking the provided functionalities. Emphasis was placed on following the provided guidelines and instructions if possible, as forking of the resource could result in errors in software's functioning by the time the next update was done to the obtained resource.

"Of course, we did not know that this [forking the resource] will break down, though we knew that it is a bit over what the SDK was able to provide, and now I would think again whether that was a wise thing to do. Better to make a request to them [resource provider] or just wait if a feature like that will be provided by them in the future" (int5).

In a similar manner, it was advisable to make the connections to the integrated resource loose, as tight integrations could lead to problems.

"If the connection is very tight between systems […], then whatever change in one system will create problems in the other, so it would be ideal if both systems could maintain their relative independence and allow each of them to do their own development" (int17).

In terms of external actors, measures could be taken to avoid the harmful impacts and counter the challenges resulting from reduction of control stemming from reliance on partners and subcontractors. One relatively straightforward way of doing this would be to rely on partners and companies that one already knows and has relatively good relations with, or otherwise has a good reputation. Contractual factors and regulations such as General Data Protection Regulation (GDPR) have also established certain commonly agreed upon guidelines and have thus helped to counter reduction of control. In order to reduce the uncertainty even further, the ability to test without committing oneself fully was also seen as useful.

"One solution is to do a proof-of-concept before the final decision, so you don't commit yourself before making sure that the resource is the right one" (int10).

Occasionally, local actors were preferred, as they were viewed as being more aware of the local context and business processes. Similarly, smaller local actors were sometimes seen as giving more importance to their partners and customers; however, larger players were mentioned as being more reliable and trustworthy because they had more resources available. It was also considered important to view partners and projects from a long-term perspective instead of one-off encounters.

It is noteworthy that the interviewees rarely mentioned having back-up plans in case a resource or actor proved to be inept for the purposes of the developed software. The idea seemed to be more that once something was decided on, it was quite difficult and costly to do away with those resources or partners and switch to others. As a result, if problems occurred, the general thinking seemed to be to deal with challenging situations as they emerged and not spend too much effort trying to prepare for those beforehand by making, for example, concrete back-up plans.

"There is a bit of that type of thinking [having back-up plans], but I feel other options are not really thought of that much, and if problems appear, then those need to be fixed with the resources available, or then start thinking if there is another way to get the data required." (int17)

5. Discussion

Reduction of control for an individual software company results from the move toward a more networked development environment, which emerges from reliance on external technological resources as well as partnerships with other actors. The ability to count on external resources and actors provides the companies multiple benefits, but as those benefits also lead to diminishing control over the developed software, particular challenges and risks also surface. These challenges have negative implications in terms of the predictability and stability of software projects and need to be mitigated in some form or another. Based on our findings, two principal strategies most often emerge, as the companies in their software development either turn inwards or then seek to strengthen the overall system that enables the networked operating environment to function. These strategies are not mutually exclusive but often interlinked, since resorting to one strategy tends to diminish the need to adopt the other one.

The first strategy, turning inwards, is simply trying to maintain control over the software under development by doing as much as possible in-house. Instead of being binary, the decision on building software in-house vs. using external resources and actors should be viewed more as a continuum. In this continuum, companies decide what is the suitable amount of control that they wish to have. On the one extreme of retaining control are practices such as building many of the functionalities within the company without resorting to external resources or partners. When moving along the continuum, some control is forfeited as software companies utilize external technological resources, but those resources do not have a substantial role in the software's functioning, there are alternatives available for the resources, or in the case of external actors, they have more of a role as subcontractors with clear hierarchical structures. Toward the other end of the continuum, companies are having less and less control over software development as they increasingly resort to externally provided technological resources and partnerships, and as a result, have few means to impact decisions that are made externally even though those decisions may considerably impact the functioning of their software.

This is where the second strategy, that is, system strengthening, begins to gain more ground as its focus is on seeking predictability and stability on the system level. To compensate for the reduction of control, different practices can be applied, such as avoiding excessive forking, making sure support is available, or relying only on partners with proven track records. In addition to these, stability and predictability are sought from regulation but primarily by relying on established standards, protocols, and common frameworks and tools. The foundation of these practices is more on the systemic level, as the aim is to create predictability and stability in how the external resources operate and impact the company's own software. Overall, this

second strategy focuses on finding alternative sources for stability when those cannot be achieved by developing everything in-house. Linked to this, reliance on an open source that allows more transparency in terms of the acquired resource is valued more. On the organizational side, similar practices can be observed, as the partners in projects should abide by the same set of standards and utilize established tools for communication and information sharing. Reliance on well-known actors as well as legal frameworks compensate for the loss of stability and predictability following that.

In other words, as control over the developed software is reduced due to the reliance on external resources and actors, this can be compensated for by aiming to bring stability and predictability to a system level where each of the software development companies operates. If those two factors, stability and predictability, can be obtained on a system level, this further contributes toward the increasing utilization of external resources and partners. In software development, this would further enable, for example, the loosening of vertical operating models that focus on developing software in-house. Similarly, having a stable and predictable operating environment will strengthen the position of the type of non-focal actors discussed by Selander et al. [6] and allow more room for the smaller actors that function as resource takers to operate in.

Two research areas are of importance regarding this in terms of future studies. The first one evolves around looking at the implications of these developments regarding notions such as generativity. By adhering to strict standards and utilizing the provided resources only as they are intended, this may also lead to a reduction in the ways different resources can be utilized, and with that, possibilities for companies to differentiate themselves from one another and gain competitive advantage from software. However, this may be contrasted, for example, by the number of resources available.

Second, the question remains about how far these strategies and practices that seek predictability and stability from the system-level are those of the weak, and if the resource providers and bigger actors such as platform and cloud infrastructure owners have interest in promoting stability on a system-level or if they see those as leading to reduction of the control they currently possess. The situation might present itself differently when the power balance is on one's side, that is, with the actor able to impact others by its decisions and functions more as a norm-giver instead of a taker. Overall however, it could also be argued that having a relatively stable and predictable operating environment would benefit all of the actors, no matter their size or position.

6. Conclusion

This paper has examined how software development that takes place in a networked operating environment tries to balance the loss of control by utilizing particular strategies and practices. Two main strategies were identified: either turning inwards and developing more in-house, or alternatively, seeking stability and predictability on a system level as well as strengthening the system and, in that way, mitigating the loss of control. The key questions that need further exploration are whether these practices inhibit generativity and to what extent those strategies and practices are engaged in by those who find themselves in a relatively weak position vis-à-vis resource providers and other more powerful actors in the environment.

7. References

[1] V. D. Bianco, V. Myllärniemi, M. Komssi, and M. Raatikainen, "The Role of Platform Boundary Resources in Software Ecosystems: A Case Study," in *2014 IEEE/IFIP Conference on Software Architecture*, Apr. 2014, pp. 11–20.

[2] K. Karhu, R. Gustafsson, and K. Lyytinen, "Exploiting and Defending Open Digital Platforms with Boundary Resources: Android's Five Platform Forks," *Info. Sys. Research*, vol. 29, no. 2, pp. 479–497, Jun. 2018.

[3] S. Jansen and M. A. Cusumano, "Defining software ecosystems: a survey of software platforms and business network governance," *Software Ecosystems*, Apr. 2013, Accessed: Jun. 12, 2020. [Online]. Available: https://www.elgaronline.com/view/edcoll/9781781955628/9781781955628.00008.xml.

[4] D. Tilson, K. Lyytinen, and C. Sørensen, "Research Commentary—Digital Infrastructures: The Missing IS Research Agenda," *Information Systems Research*, vol. 21, no. 4, pp. 748–759, Nov. 2010.

[5] A. Gawer, "Bridging differing perspectives on technological platforms: Toward an integrative framework," *Research Policy*, vol. 43, no. 7, pp. 1239–1249, Sep. 2014.

[6] L. Selander, O. Henfridsson, and F. Svahn, "Capability Search and Redeem across Digital Ecosystems:," *Journal of Information Technology*, Sep. 2013, Accessed: Apr. 09, 2020. [Online]. Available: https://journals.sagepub.com/doi/10.1057/jit.2013.14.

[7] B. Eaton, S. Elaluf-Calderwood, C. Sørensen, and Y. Yoo, "Distributed Tuning of Boundary Resources: The Case of Apple's iOS Service System," *MIS Quarterly*, vol. 39, no. 1, pp. 217–243, 2015.

[8] Y. Yoo, K. Lyytinen, and R. J. Boland, "Distributed Innovation in Classes of Networks," in *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)*, Jan. 2008, pp. 58–58.

[9] C. Y. Baldwin and C. J. Woodard, "The Architecture of Platforms: A Unified View," in *Platforms, Markets and Innovation*, A. Gawer, Ed. Cheltenham, UK: Edward Elgar Publishing, 2009.

[10] K. H. Rolland, L. Mathiassen, and A. Rai, "Managing Digital Platforms in User Organizations: The Interactions Between Digital Options and Digital Debt," *Information Systems Research*, vol. 29, no. 2, pp. 419–443, May 2018.
[11] Y. Yoo, O. Henfridsson, and K. Lyytinen, "Research Commentary—The New Organizing Logic of Digital Innovation: An Agenda for Information Systems Research," *Information Systems Research*, vol. 21, no. 4, pp. 724–735, Nov. 2010.
[12] I. Richardson, V. Casey, J. Burton, and F. McCaffery, "Global Software Engineering: A Software Process Approach," in *Collaborative Software Engineering*, I. Mistrík, J. Grundy, A. Hoek, and J. Whitehead, Eds. Berlin, Heidelberg: Springer, 2010, pp. 35–56.
[13] R. Sangwan, M. Bass, N. Mullick, D. J. Paulish, and J. Kazmeier, *Global Software Development Handbook*. CRC Press, 2006.
[14] P. Wagstrom, C. Jergensen, and A. Sarma, "Roles in a Networked Software Development Ecosystem: A Case Study in GitHub," *CSE Technical reports*, Jan. 2012, [Online]. Available: https://digitalcommons.unl.edu/csetechreports/149.
[15] A. Ghazawneh and O. Henfridsson, "Balancing Platform Control and External Contribution in Third-Party Development: The Boundary Resources Model," *Information Systems Journal*, vol. 23, no. 2, pp. 173–192, Mar. 2013.
[16] W. Venters and E. A. Whitley, "A critical review of cloud computing: researching desires and realities," *J Inf Technol*, vol. 27, no. 3, pp. 179–197, Sep. 2012.
[17] P. C. Evans and R. C. Basole, "Revealing the API ecosystem and enterprise strategy via visual analytics," *Commun. ACM*, vol. 59, no. 2, pp. 26–28, Jan. 2016.
[18] Y. Yoo, K. Lyytinen, B. V. Thummadi, and A. Weiss, "Unbounded Innovation with Digitalization : A Case of Digital Camera," presented at the 2010 Annual Meeting of the Academy of Management, 2010.
[19] K. Ulrich, "The role of product architecture in the manufacturing firm," *Research policy*, vol. 24, no. 3, pp. 419–440, 1995.
[20] C. Y. Baldwin and K. B. Clark, *Design Rules: The power of modularity*. London: MIT Press, 2000.
[21] F. Salvador, "Toward a Product System Modularity Construct: Literature Review and Reconceptualization," *IEEE Transactions on Engineering Management*, vol. 54, no. 2, pp. 219–240, May 2007.
[22] E. von Hippel, "Task partitioning: An innovation process variable," *Research Policy*, vol. 19, no. 5, pp. 407–418, Oct. 1990.
[23] S. K. Fixson, "Product architecture assessment: a tool to link product, process, and supply chain design decisions," *Journal of Operations Management*, vol. 23, no. 3, pp. 345–369, Apr. 2005.
[24] M. Jacobides, J. P. MacDuffie, and C. J. Tae, "Agency, structure, and the dominance of OEMs: Change and stability in the automotive sector," *Strategic Management Journal*, vol. 37, no. 9, pp. 1942–1967, 2016.
[25] J. P. MacDuffie, "Modularity-as-Property, Modularization-as-Process, and 'Modularity'-as-Frame: Lessons from Product Architecture Initiatives in the Global Automotive Industry," *Global Strategy Journal*, vol. 3, no. 1, pp. 8–40, 2013.
[26] N. Argyres and L. Bigelow, "Innovation, Modularity, and Vertical Deintegration: Evidence from the Early U.S. Auto Industry," *Organization Science*, vol. 21, no. 4, pp. 842–853, 2010.
[27] O. Henfridsson and Y. Yoo, "The Liminality of Trajectory Shifts in Institutional Entrepreneurship," *Organization Science*, vol. 25, no. 3, pp. 932–950, 2014.
[28] M. Jacobides, C. Cennamo, and A. Gawer, "Towards a Theory of Ecosystems," *Strategic Management Journal*, vol. 39, no. 8, pp. 2255–2276, 2018.
[29] K. J. Sullivan, W. G. Griswold, Y. Cai, and B. Hallen, "The structure and value of modularity in software design." Association for Computing Machinery, Sep. 01, 2001, Accessed: Apr. 09, 2020. [Online]. Available: https://doi.org/10.1145/503271.503224.
[30] S. Nambisan, K. Lyytinen, A. Majchrzak, and M. Song, "Digital Innovation Management: Reinventing Innovation Management Research in a Digital World," *Management Information Systems Quarterly*, vol. 41, no. 1, pp. 223–238, Mar. 2017.
[31] O. Hanseth and K. Lyytinen, "Design Theory for Dynamic Complexity in Information Infrastructures: The Case of Building Internet," *J Inf technol*, vol. 25, no. 1, pp. 1–19, Mar. 2010.
[32] A. Tiwana, "Does technological modularity substitute for control? A study of alliance performance in software outsourcing," *Strat. Mgmt. J.*, vol. 29, no. 7, pp. 769–780, Jul. 2008.
[33] D. Tilson, C. Sørensen, and K. Lyytinen, "Change and Control Paradoxes in Mobile Infrastructure Innovation: The Android and iOS Mobile Operating Systems Cases," in *2012 45th Hawaii International Conference on System Science (HICSS)*, Jan. 2012, pp. 1324–1333.
[34] A. Tiwana, *Platform Ecosystems: Aligning Architecture, Governance, and Strategy*. Newnes, 2013.
[35] M. de Reuver, C. Sørensen, and R. C. Basole, "The digital platform: a research agenda," *Journal of Information Technology*, vol. 33, no. 2, pp. 124–135, 2018.
[36] A. Dubois and L.-E. Gadde, "Systematic combining: an abductive approach to case research," *Journal of Business Research*, vol. 55, no. 7, pp. 553–560, Jul. 2002.