8-14-2019

Making Digital Infrastructures More Generative Through Platformization and Platform- driven Software Development: An Explorative Case Study

Kathrine Vestues
Norwegian university of science and technology, kathrine.vestues@ntnu.no

Rolland Knut
University of Oslo, knutr@ifi.uio.no

MAKING DIGITAL INFRASTRUCTURES MORE GENERATIVE THROUGH PLATFORMIZATION AND PLATFORM-DRIVEN SOFTWARE DEVELOPMENT: AN EXPLORATIVE CASE STUDY

Research paper

Vestues, Kathrine, Norwegian university of science and technology, Norway, kathrine.vestues@ntnu.no

Rolland, Knut, University of Oslo, Norway, knutr@ifi.uio.no

Abstract

Digital innovation platforms are particularly known for their generativity producing and reproducing flexible solutions for multiple user groups and attracting third-party developers. Consequently, as a concept, generativity often relates to the consumer market as its focal unit of analysis. In contrast, this paper takes a public sector organization as a unit of analysis, investigating the potential for developing more generative digital infrastructures through processes of platformization and platform-driven software development practices. We define platformization as the transition from silo-based organization to platform-based organization, where the dismantling of monolithic legacy systems enables new ways of organizing the development and maintenance of software development. By decoupling systems, organizations can recouple their organization to overcome the limitations of traditional silo-based organization. Our contribution is twofold. First, we contribute by describing platformization as a sociotechnical process that consist of changes to both infrastructure and organization, where the dismantling of monolithic applications enables new ways of organizing the development and maintenance of software. Second, we contribute by theorizing how platformization processes can produce increased generativity for the IT organization and its digital infrastructure.

Keywords: Platformization, Platform-driven software development, Digital platforms in public sector organizations, Digital infrastructures.

1 Introduction

Digital platforms have recently attracted much attention by both practitioners and IS scholars. For practitioners, digital platforms have become the latest buzzword for establishing new business and rapidly scaling services to a large number of consumers. In the IS literature, digital platforms are often portrayed as multi-sided markets enabled by flexible platform architectures that are capable of producing high variety as well as high quality of products and services (Gawer and Cusumano 2014; Parker et al. 2016; Tiwana 2014; Wareham et al. 2014; Yoo et al. 2010). As such, digital platforms are often put forward as technologies with an almost inherent capacity for continuously producing new features and innovations. For example, recent IS research has investigated how innovation platforms like Google Android and Apple iOS rapidly evolve through provision of 'boundary resources' that balance platform control and - at the same time, a certain level of flexibility and openness for attracting third-party developers (Eaton et al. 2015; Ghazawneh and Henfridsson 2013). Such insights underscore that digital platforms are anything but stable and discrete technologies. They continuously evolve in terms of changing sociotechnical arrangements of functionality, architecture, governance regimes, and involved actors. Although digital platforms come in many technical forms and organizational arrangements (Gawer and

Cusumano 2014), their layered modularized architecture (Rolland 2018; Yoo et al. 2010) and distributed organization (Yoo et al. 2012), make them *distinctly different* from traditional IT-systems and IS in organizations. Following IS literature, a key explanation for the widespread adoption of digital platforms (and the associated ecosystems), are their *generative* dynamics (e.g. (Constantinides et al. 2018; de Reuver et al. 2017; Henfridsson and Bygstad 2013)). As such, generativity is defined as the "*capacity to produce unanticipated change through unfiltered contributions from broad and varied audiences*" (Zittrain 2008). Thus, it is suggested that platforms as digital artifacts evolve in a recursive manner; the platform itself is used as a resource for evolving the platform further. It is generally suggested that modularized and open digital solutions are more generative than less modularized and more closed ones.

Generativity is also a concept used in relation to digital (information) infrastructures (Monteiro et al. 2013). Here generativity is problematized and related to strategies, practices and designs that can make digital infrastructures more generative in order to more smoothly establish or scale new infrastructures. We define digital infrastructure as the collection of technological and human components, networks, systems, and processes that contribute to the functioning of an information system (Tilson et al. 2010a).

Often, large public organizations have a fragmented digital infrastructure consisting of legacy systems as well as more modern applications and services developed on multiple technology platforms. Scandinavian IS scholars have a particular long track record in researching digital (information) infrastructures. This stream of IS research has emphasized various solutions and conceptualizations, as for example, *cultivation* (Aanestad and Jensen 2011; Ciborra 2000), *gateways* (Hanseth 2001), and *bootstrapping* by simplicity and usefulness of early designs (Hanseth and Lyytinen 2010) for solving problems related to the inertia of installed base typically consisting of a wide variety of more or less interconnected legacy systems (Grisot et al. 2014; Hepsø et al. 2009). More recently, platformization has been suggested as a viable way of increasing flexibility of digital infrastructures by increasing their modularity (Bygstad and Hanseth 2018; Islind et al. 2016). With their modular architecture, platforms allow applications to be added, changed or remove without affecting other applications and modules. For most organizations such a strategy will however require a gradual transition - where legacy systems are gradually dismantled into modular applications (Bygstad and Hanseth 2018).

Against this backdrop, we aim to examine how software platforms allow organizations to change the way they develop and maintain software and thereby increase generativity and innovation. The mediating capacity of platforms has enabled a new way of developing and offering services and has attract attention from both researchers and practitioners. This capacity is however not restricted to open platforms and external developers. Organizations can equally well use *internal* platforms to coordinate the efforts of autonomous teams. The modular layered architecture of the platform allows cross-functional teams to develop and deploy applications without coordinating their efforts with other teams and applications. This potentially enables generativity which is difficult in fragmented digital infrastructures consisting of a web of interconnected legacy systems. Based on the transformative potential of internal software platforms we ask the following two research questions:

1. *What characterizes platformization in organizations?*
2. *How can platformization contribute to increased generativity within organizations?*

To further our understand of how digital platforms contribute to generativity within organizations, we draw on insights from the service dominant logic (Lusch and Nambisan 2015). Here, generativity is said to increase through increased availability of resource (skills, knowledge, or software), and by establishing activities and processes that make it easy for actors in the organization to use and combine these resources.

Empirically, we draw on findings from a longitudinal case study of a large public service organization. In 2017, the organization changed its software delivery strategy: Silo-organization and staged delivery model was replaced by cross-functional teams and continuous deliveries. This change was enabled by the platformization of the digital infrastructure: Monolithic applications were gradually reimplemented as modular applications, allowing development teams to work more autonomously.

Our contribution is twofold. First, we contribute by giving rich empirical description of the platformization process and especially how platform-driven software development practices involving an application platform enabled rapid scaling of new applications despite existing installed base of legacy systems. In other words, we show how platformization increased the generativity of an existing infrastructure characterized by inertia and path-dependency.

Second, we contribute to the platform literature by suggesting the concepts of 'decoupling' and 'recoupling' as a way of theorizing platformization as a socio-technical process.

We organize the remainder of the paper as follows. First, we introduce related literature on platforms and platformization. We then present our research method, before continuing with our results. Finally, we discussion our findings in relations to our theoretical framework, before making some concluding remarks.

2 Related literature

2.1 Generativity and platforms

Generativity can be defined as system's ability to produce unprompted change (innovation) through contributions from uncoordinated audiences (Zittrain 2008; Zittrain 2006). More precisely, the generativity of a systems is determined by its ability to leverage across a range of tasks, adaptability to a range of different tasks, and ease of mastery and accessibility. We view the "system" in this context to be the combination of social actors and technology within an organization. This means that generativity describes not only the objective characteristics of the system, but the relation between the system and its actors, and the relation among actors.

Generativity and innovation are closely related; The more generative a system, the higher the likelihood of innovation. In its most fundamental form, all innovation can be seen as the recombination (Henfridsson et al. 2018; Yoo et al. 2010).

To explain how digital innovation unfolds, we draw on insights from service-dominant logic (Lusch and Nambisan 2015). Service-dominant logic emerged as a reaction to the goods-oriented logic which has focused on innovation relating to tangible products. Service-dominant logic removes the divide between tangible and intangible products, and views innovation as an unbounded process where actors recombine resources that might be both tangible and intangible. Examples of intangible resources are skills, knowledge, and digital components. Service-dominant logic also differs from goods-oriented perspectives in its view on producers and consumers. While actors have traditionally been considered either consumers or producers of services, the service-dominant logic acknowledges that an actor can both consume and produce innovation. This is particularly relevant for digital innovation. Digital resources are editable, reprogrammable and redistributable (Kallinikos et al. 2013), and can be combined and recombined in ways that tangible resources cannot. Actors in digital innovation will both consume and produce services – sometimes as part of the same transaction.

To increase innovation, organizations must increase the availability of resources, and the ease with which these resources can be recombined (Lusch and Nambisan 2015). The availability of resources can be described in terms of "resource density", where resource density denotes the ability to mobilize contextually relevant resource and effective and efficient way. The recombination of resources is described as a process of "resource integration".

Software platforms have become a popular element in digital innovation processes. Both because their modular architecture makes them easy to extend and combine, and because of their mediating abilities.

Although platforms come in many shapes and forms, most platforms share a common architecture based on reuse of core components to achieve economies of scale and create a wide variety of complementary components (Baldwin and Woodard 2009). In this paper, we view the platform as a "modular structure that consists of tangible and intangible components (resources) and facilitates the interaction of actors and resources (or resource bundles)" (Lusch and Nambisan 2015).

Platforms present and ideal type of generativity through their modular architecture, and strategies for profiting from this generativity has caught the attention of numerous researchers (cf. (Bygstad 2015; Tilson et al. 2010b)). By opening up the platform to external developers, the platform owner can draw on the generative capacity of unlimited numbers of third-party developers. A key concern in the platform literature has therefore been strategies for attracting contributors and increasing generativity (see for instance (Boudreau 2010; Boudreau 2012; Wareham et al. 2014)). An underlying premise in much of this literature has however been that platforms must be open to third party contributors if the platform owner is to benefit from the generative capacity. As phrased by Yoo et al. (2010: "Although it is theoretically possible to pursue such generativity within the boundaries of a single firm or its existing supplier network, a firm's ability to do so on a practical basis is limited by its economic, structural, cognitive and institutional constraints".

For obvious reasons, not all firms are willing or able to open their platforms. They wish to increase generativity by using internal platforms (Gawer and Cusumano 2014). We therefore need strategies where platforms can be used to increase resource density and improve resource integration within the boundaries of a single firm. To understand how this can be achieved, we will explore the concept of "continuous software development", and how it can transform the way large-scale software organizations develop and manage software.

2.2 Platform-driven software development

Ever since its inception, the software industry has struggled with the complexity of software development. The need to manage complexity has initiated a series of strategies and methods. These range from up-front specification and staged development, to more incremental approaches where activities such as specification and development are interleaved. By performing specification, development, and testing iteratively, requirements can be gradually elaborated and reprioritized as knowledge about business domain and technology matures. The rationale behind these approaches is that by deferring decisions, uncertainty and risk is reduced.

Iterative development methods have long traditions (Larman and Basili 2003), but were popularized through the introduction "agile" methods in the 2000's (Dingsøyr et al. 2012). Agile methods are a collection of practices based on input from experience practitioners. They came as a reaction to traditional plan-based methods, which emphasized a "rationalized, engineering-based approach" (Dyba 2000). As opposed to plan-based approaches, agile methods favor close collaboration between business experts, developers and users, enabling constant feedback and learning in so called "cross-functional" teams.

Recent trends extend these thoughts: Rather than focusing on agile methods by themselves, the practices are seen in relation to activities in other parts of the organization. This cross-organizational focus has brought about concepts such as DevOps (Humble and Molesky 2011) and BizDevOps (Gruhn and Schäfer 2015). We refer to these trends as "continuous software development" (Fitzgerald and Stol 2017), where software is developed and managed by cross-functional teams that specify, prioritize, develop, release software continuously. These practices remove harmful disconnects between business, development, and operations, and ensure continuous feedback and learning.

The disintegration of traditional organizational boundaries has long been recognized as an enabler for learning and innovation (Brown and Duguid 2001), where collaboration practices increase resource integration by challenging knowledge boundaries (Lusch and Nambisan 2015). However, these continuous and boundary spanning development practices require team autonomy and are therefore best suited for small-scale projects and with few developers and limited scope.

In large-scale software development projects software will be integrated, operated, and maintained in connection with existing systems (Pipek and Wulf 2009; Tilson et al. 2010a), where the existing installed base must be taken into account (Aanestad and Jensen 2011; Hepsø et al. 2009). Due to the large number of dependencies, attempts at scaling agile and continuous development practices in large-scale settings

has proven difficult. In complex network of systems, components and stakeholders, changes to one system will affect other stakeholders and systems. The consequent need for coordination and control makes continuous development practices difficult.

Hanseth and Lyytinen (2010 argue that development must build on existing solutions, and suggest modularization as a strategy to develop new features in an evolutionary manner. Aanestad and Jensen (2011 argue that software development not only needs to focus on modularization of the software itself – but also on modular implementation strategies that mobilize stakeholders. Building on these insights, we conclude that software development in large-scale settings will profit from modular implementation strategies that involve representatives from all parts of the development pipeline. We propose platformization as one such strategy.

3 Theorizing platformization

Several authors have used the term "platformization": Helmond (2015 uses the concept to describe the rise of the social web, while Islind et al. (2016 uses the term to describe the emergence of a platform ecosystem. Bygstad and Hanseth (2018 describe platformization as "a process where IT silo solutions are gradually transformed to a platform-oriented digital infrastructure". We adopt this latter view. However, for organizations to profit from the generativity of a platform architecture, the transformation has to be accompanied by organizational changes (Yoo et al. 2010). We therefore extend our conceptualization of platformization to include organizational changes enabled by the modular architecture of the platform.

Hence, in this paper, we view platformization as a sociotechnical process that consist of changes to both infrastructure and organization, where the dismantling of monolithic applications enables new ways of organizing the development and maintenance of software. More specifically, we see platformization to unfold through the separate but related processes of decoupling and recoupling, where "decoupling" refers to dismantling of legacy systems into modular applications, and "recoupling" refers to a restructuring of development practices such that the organization can profit from the generativity of the platform architecture. The objective of each of these processes is to increase generativity by increase resource density and improved resource integration.

Our theoretical framework is summarized in the table below.

Concept	Description	Relevance
Generativity	Denotes "a system's overall capacity to produce unprompted change driven by large, varied, and uncoordinated audiences" (Zittrain 2008)	By altering the organizing logic of software development, firms can increase the generativity of internal software development
Platformization	The socio-technical process of transition from silo-based organization to platform-based organization of software development. Unfolds through processes of "decoupling" and "recoupling"	The dismantling of monolithic applications enables new ways of organizing the development and maintenance that can increase the generativity inside software development organizations
Decoupling of systems	Process of establishing an appropriate modular architecture that enhances resource density Nambisan (Lusch and Nambisan 2015).	By establishing a platform architecture, development teams can make changes to applications without affecting other parts of the system. This reduces the need for coordination and control
Re-coupling of organization	Process of establishing practices and roles that facilitate resource integration (Lusch and Nambisan 2015).	Removes harmful disconnects between activities such as planning, development and implementation

		(Fitzgerald and Stol 2017). Increases learning and innovation through disintegration of organizational boundaries (Brown and Duguid 2001)

Table 1. Theoretical concepts defined

4 Method

4.1 Case description

The fieldwork was conducted within the IT department of a large public welfare organization in a Scandinavian country. The organization was established in 2006 following the merger of the formerly independent National Insurance Agency, Employment agency, and the locally controlled Social Services. The organization has 19,000 employees and provides welfare solutions for close to 800,000 active users.

The IT department has approximately 700 employees and 400 consultants and maintains and operates 300 applications. The application portfolio is made up of several generations of solutions, from mainframe solutions, to newer web-oriented applications, as well as standard systems that support operations such as accounting, payroll and document production.

In addition to securing income through schemes such as age pension and disability benefit, the organization is responsible for mobilizing and improves the population´s work ability. To increase the efficiency and quality of services, they have begun to digitize services and automate manual application processing. With a growing number of digital solutions and interdependencies between solutions, the digital infrastructure was however becoming difficult to change and maintain. To address this problem, the organization has embarked on a platformization strategy: Silo systems are decoupled into modular applications, and the hierarchical organization recoupled into cross-functional teams that perform continuous software development. As part of this process, the organization has changed its sourcing strategy. After years of outsourcing, they are insourcing the development and maintenance of core products.

4.2 Data collection and analysis

This paper reports from a longitudinal case study (Pan and Tan 2011; Yin 2009). Data was collected through in-depth semi-structured interviews (Myers 2013), observations, and document analysis. A total of 40 interviews were conducted (Table 2), of which 23 have been transcribed. Interviews lasted 45 - 60 minutes. Data was collected both from the line-organization (section for IT Architecture), and from ongoing projects. Data collection started in February 2017 and was completed in April 2019.

Roles	Parental Benefit project	IT architecture	Total
Managers	14	1	15
Developers	11	3	14
Architects	10	1	11

Table 2. List of interviews.

The data analysis was iterative and overlapped with data collection, thus granting flexibility to respond to emergent themes (Eisenhardt 1989). This followed Pan and Tan's (2011) process approach, with a framing cycle, followed by an augmenting cycle. During the framing circle, we performed our initial interviews and identified the ongoing transformation.

Through our data analysis, we found that changes to the organization of software development were preconditions by changes to the digital infrastructure and the introduction of software platforms. We

therefore decided to investigate the mutual shaping of platform and organization. Since mutual shaping is difficult to capture, we decided to temporally bracket events relating to the introduction of application platforms and to changes to the organization of software development (Langley 1999). This gave us two separate but related processes.

To understand how these changes affected generativity, we iterated between our data and relevant literature. Based on these insights we constructed and extended out theoretical framework. Events relating to the establishment of the platform indicated a gradual decoupling of technical infrastructure. We labelled this a process of "decoupling". As technical systems (applications) became increasingly decoupled, the need for coordination between development teams was reduced. This enabled increased team autonomy and the establishment of cross-functional teams. We labelled the establishment cross-functional teams and continuous development practices a process of "recoupling".

Error! Reference source not found. visualizes the progression of events relating to the two processes. It shows that although the initial decoupling of technical infrastructure preceded the organizational change, the processes were overlapping.

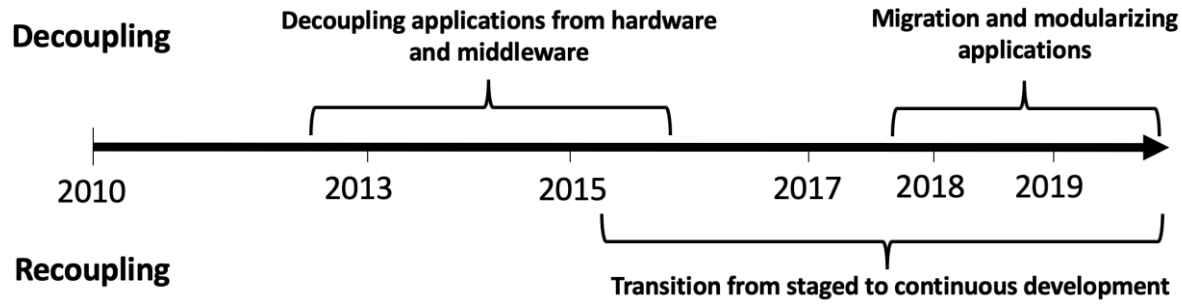

Figure 1. Platformization unfolds through interrelated processes of decoupling and recoupling

5 Results

In this section we present the processes of recoupling and decoupling. Although conceptually distinct, the processes mutually shape each other and overlap in time.

5.1 Decoupling legacy systems (2012 - ongoing)

The decoupling of systems begun in 2012 and was done in two stages: First, applications were separated from the technical infrastructure. Second, legacy systems were rewritten as modular applications.

The first stage began in 2012, when the welfare organization introduced its first application platform. The application platform was based on virtual machines running JBoss applications servers on a RedHat Linux operating system. Most existing systems were migrated to the application platform.

The introduction of application platform affected development and operations in several ways. Most importantly, through the standardization of technical infrastructure. Before the application platform was established, systems ran on dedicated physical servers that were specified by projects. This resulted in a range of different technologies - depending on the competence and preference of the given project. The application platform reduced technical variety and relieved projects of tasks relating to technical infrastructure. It also reduced hardware costs as physical servers were replaced by virtual machines. One informant described the change in the following way:

"Let me give you an example. The first project I worked on was in 2005-2006. We spent quite a bit of time deciding which application server the solutions should run on, and which operating system the application server should run on. [...] Projects spent a long time figuring out whether to use Windows

or Linux, or whether to use a mainframe machine. [...] Projects don't do that anymore. Because now everything is standardized".

The application platform semi-automated provisioning of servers and deployment of software. Provisioning time was reduced from weeks to minutes, and software could be automatically configured and deployed. This simplified operations, and reduced error rates. The introduction of the application platform, and the subsequent migration of applications, was later described as the Operations department's "great boost". It increased the predictability and stability of operations.

But although the application platform separated applications from hardware and middleware, applications were still large and tightly coupled. Software deployments therefore had to be carefully coordinated. This was done by bundling deployments into 4 yearly releases. As described by one informant:

"The size of the coordinated releases could approach 150 000 development hours with risks to both technical and project management".

Although effective in reducing errors, the required coordination effort, combined with reduced deployment rates, proved expensive and inflexible. It could take months from a feature was developed until it was deployed. To increase the frequency and flexibility of the deployment process, dependencies between systems had to be reduced. This led to the second stage of the decoupling process, which began in 2017.

In the second stage of decoupling, the organization began to dismantle legacy systems. To simplify this process, the organization introduced a second-generation application platform. The new application platform was built on Kubernetes, an open source framework based on container technology. A member of the platform team described the framework in the following way:

"Kubernetes is open source framework developed by Google. It contains all of Google's experience with infrastructure over the past 15 years - re-written by the same people. It is in a way to take the world's best operations person and automate him completely. That's what Kubernetes is. It provides a lot of tools for running in production, making it more robust, scalable, and all kinds of things".

Applications were gradually migrated from the old to the new application platform, and as applications are migrated, legacy systems are split into smaller applications. One of the developers explains the second stage of the decoupling process as follows:

"What is happening is that one application on the old platform becomes multiple applications on the new platform. This is because applications are split up as they are migrated. This is typical. An old application becomes many new applications".

In addition to applications running on the application platform, the organization had a number of large legacy systems that were not running on the first-generation application platform. They had been too tightly coupled to the hardware they were running on. These systems were gradually re-written and deployed on the new application platform.

The decoupling of applications meant that applications could be deployed independently. As explained by one of the developers:

"[With the modular architecture] you get smaller code bases. This means I can roll out my code regardless of what other teams have done".

Whereas the Operations department had been responsible for deploying applications on the first-generation platform, the second-generation platform was open to developers. To ensure that this openness did not affect the stability of systems, the organization introduced the "white-list". A white-listed application could be deployed independently, without involving Operations. Application were white-listed they had few dependencies and were low in complexity.

The modular architecture of the second-generation application platform, combined with the white-listing of applications, enabled continuous software development practices. We describe the process of establishing these practices as "recoupling".

5.2 Recoupling the organization (2015 - ongoing)

Up until 2017, most software development projects were organized as a staged workflow with defined phases and hand-overs. As described by one employee:

"The first phases consisted of requirement specification in Epics and refinement in user stories […], analysis and design, culminating in a detailed release plan based on carefully estimated user stories. These phases were performed by employees consisting of experts from business domains […] getting assistance from other experts when needed. Typically, a project was started for new development or major modifications and […] organized by product owners in a prioritized queue with virtual backlogs for each team […]. A project could involve 10 – 20 teams with dependencies to be coordinated. Software development was outsourced to consulting firms and handled through a series of time boxed iterations […] involving employees for reviews, acceptance tests and delivery. The development teams executed the iterations with the same length and in the same rhythm to ease handling of dependencies".

The transition from plan-based to continuous software development began in 2015 with the piloting of a new software delivery method. In the new delivery model, cross-functional teams would work in close collaboration with users releasing software incrementally and frequently. A team consisted of developers, tester and business experts. This recoupling removed boundaries between different functions and departments. While the decoupling of systems had provided technical autonomy, the establishment of cross-functional teams provided organizational autonomy. With business experts and developers in close approximation, the team could specify and re-prioritize without involving other parts of the organization. One of our informants described the new team structure in the following way:

«The project was initially set up with a single team with less than ten members comprised of two UX-designers, two subject matter experts, one software architect, one developer, operations support and agile coaches».

A major motivation behind the transition towards continuous development was abolishment of handovers:

"So that's been the main goal really: That the developer should be involved from conception of an idea to, yes really until we turn off the system. And there should be no handovers. So, not first service design, then architecture, then solution architecture, then the developer, then the tester, then operations. It should be the developer all the way".

Although the project only delivered a stand-alone application with few dependencies to other parts of the organization, it proved the efficiency and viability of the delivery method. In 2016, the organization therefore decided to apply the delivery method to other projects and products in the organization. Even the largest of the development projects were in 2018 reverted to the novel delivery model. What had begun as a staged development process with upfront planning and coordinated releases was half-way through the project reorganized as cross-functional teams working on modular applications.

Like many public organizations, WelfareOrg had since their inception in 2006 outsourced software development and maintenance to consultant companies. As part of the transition towards continuous development, this sourcing strategy was changed. They would assume responsibility for the development *and maintenance of core systems:*

"As of 2017 a new sourcing strategy was developed, where all core development capacity and knowledge should be insourced, and only extra capacity to handle capacity peaks should be outsourced. New contract types are also under evaluation. Previously, most contracts were of type target price dividing the risk between the client and the vendor" (Manager)

With the new sourcing the organization would take ownership to its own products. It had proven advertently difficult to coordinate development and maintenance across multiple suppliers which all had individual needs. As expressed by the CTO:

"It is quite difficult to get 15-20 suppliers – who all have their own interests, to work for your benefit […]. How do you coordinate them across 300 systems? How do you ensure adequate quality in the code when you are not present in the code? And that you have automated testes across solutions delivered by different suppliers? That is impossible, you know»

The modular architecture of the application platform was a precondition for the ongoing change. By decoupling applications, development teams could develop and release applications frequently, ensuring continuous feedback from systems and users.

6 Discussion

Platform generativity has been described a platform's ability to attracting large numbers of heterogeneous components and contributors (Yoo et al. 2010). Such heterogeneity is most easily achieved outside the boundaries of a firm - where platform owners can draw on the autonomous contributions of an unlimited number of third-party developers. Researchers have therefore focused on the generative capacity of open platforms, exploring strategies for attracting external contributors, and how the openness to external contributions must be balanced against the platform owners need for control (cf. (Boudreau 2010; Boudreau 2012; Wareham et al. 2014)). The underlying premise in these studies has been that economic, structural, institutional and cognitive constraints will prevent such generativity inside the boundaries of a firm. In this paper, we have challenged this premise. Although internal platforms cannot profit from the same number of contributors, we claim that organizations can achieve increased generativity through a platformization strategy. To substantiate this claim, we proceed to answer our two research questions: a) *"What characterizes platformization in organizations?"* and b) *"How can platformization contribute to increased generativity within organizations?".*

6.1 Characteristics of platformization

Platformization within organizations is characterized by the two separate but related processes of "decoupling" and "recoupling". In the decoupling process, legacy systems were dismantled into modular applications, while the recoupling processes, entailed a restructuring of the software development organization. The decoupling process proceeded in two steps: First, the application layer was separated from the underlying technical infrastructure. Second, legacy systems were dismantled into modular applications. This process was gradual and proceeded over many years: The first steps towards a platform architecture was taken in 2012. By 2019, the process was still ongoing, and would most likely continue for years to come. This resonates with other studies, which has highlighted the difficulties of replacing established systems (Svahn et al. 2017). Within large networks of interconnected systems, changes must be made incrementally (Aanestad and Jensen 2011).

In the recoupling process, the silo-based organization of software development - where distinct groups or departments were responsible for different parts of the development process, was replaced continuous development, where cross-functional teams were responsible for the entire software development cycle. This change was enabled by the modular application architecture achieved through the decoupling process. But although the decoupling of systems was a precondition for the recoupling of the organization, in practice, the two processes unfolded in parallel. The many legacy systems meant that the organization had run two software delivery processes in parallel. Interconnected legacy systems with integral architectures were managed under the traditional delivery strategy - with staged development and coordinated releases, while modular applications could be developed and managed by cross-functional teams. To ensure that the transition towards continuous development did not affect the stability of existing systems, the organization introduced the concept of "white-listing". Applications that were low in complexity and had few dependencies were put on the white-list. Only when an application had been white-listed could they be managed by a cross-functional team.

(Bygstad and Hanseth 2018) define platformization as the "process of as a process where IT silo solutions are gradually transformed to a platform-oriented digital infrastructure". They see platformization as an incremental process where the complexity of legacy systems is abstracted. This is done by establishing a set of "tools and regulations that serve as an interface between the IT silo systems and the user services" (ibid).

Our conceptualization of platformization differs from this definition in two important ways: First, we extend platformization to include both technical and organizational change. Second, in the way it approaches the installed base: While Bygstad and Hanseth (2018 describe platformization as a process of *abstracting* legacy systems, we see platformization as a process of *replacing* legacy systems. The replacement of legacy systems has been a longstanding concern among researchers and practitioners (c.f. (Feathers 2005)). We add to these discussions by viewing platformization not a technical exercise, but as a sociotechnical process where both systems and practices are replaced.

6.2 Improving generativity through platformization

To understand how platformization contributes to increased generativity inside organizations, we draw on insights from service-dominant logic (Lusch and Nambisan 2015). The ability to produce such generativity is determined by a systems resource density, and the ease with which actors in the system can recombine the resources. However, existing literature fails to acknowledge the continuous and ongoing work required to increase resource density and improve resource integration inside such organizations. To close this gap, we introduce the concepts of "decoupling" and "recoupling", where decoupling denotes the ongoing changes required to increase resource density by establishing a platform architecture, and recoupling describes the organizational changes required to improve resource integration.

In our case study, decoupling played out as a two-stage process. In the he first stage, the technical infrastructure was separated from presentation layer. In the second stage, applications were split into microserviced, increasing resource density through an increasing number of modular applications. The decoupling process, as we define it began in 2013 with the introduction of an application and was still ongoing at the time of writing. Decoupling increased resource density in three important ways: First, resource density was increased by introducing an application platform that was based on an open source framework. Although the organization only appropriated a small subset of applications and services available in the framework, this subset could continuously be changed and evolved based on the organizations need.

Second, resource density was increased by re-writing monolithic legacy systems as modular applications. Public administrations have been characterized by silo-based organization and monolithic systems that include a broad range of functional areas within the silo. For the welfare organization, this manifest itself in a few core systems that included a wide range of functional areas. The interconnectedness and complexity of the systems made them difficult and expensive to maintain. To establish a platform architecture, the systems were gradually re-written as modular applications that could run on the application platform.

Third, through the digitization of services. An increasing amount of data which had previously been paper based was implemented as services on the platform. By offering the data as reusable services, the data could then be reused by other applications and services – thereby increasing the accuracy and efficiency of transactions.

The decoupling of the technical infrastructure into a layered modular architecture allowed the organization to re-think the software delivery process. Whereas the interconnectedness of applications had necessitated coordinated releases, the decoupling meant that applications could be developed and managed independently. This enabled a recoupling of the organization and the consequent introduction of continuous software development.

To establish continuous software development, the organization made changes on multiple levels. Most important were changes to the sourcing strategy, the introduction of cross-functional teams, continuous integration, increased frequent of releases, and granting development teams access to the applications

platform. Insourcing of software development and the establishment of cross-functional teams enabled recombination of knowledge and skills that had been impossible in the plan-based delivery model.

Increased frequency of releases gave teams continuous feedback from systems and end-users, which made the team able to continuously adjust and adapt to changing user needs, making them active participants in the development process.

7 Concluding remarks

Platformization increases generativity within organizations through increased resource density and improved resource integration. Most organizations will however have legacy applications that make platformization difficult. In our research we examined one organization's platformization effort. During the one and a half years we conducted our fieldwork, the organization made radical changes to the way they planned, developed and maintained software.

The transition was, however, far from smooth. The complexity and interdependence of legacy systems make decoupling difficult. Although the organization had prepared strategies for replacing many of the legacy systems, some systems were simply too expensive to replace. The decoupling process that begun in the 2013 will therefore continue for many years to come.

Likewise, the recoupling of the organization required time and patience. This was in part due to legal concerns – contracts could not be replaced until they expired, and in part due the gradual maturing required for teams to understand what decisions they could make, and what decisions they could not.

In addition, such a fundamental change is bound to meet resistance. Although the transition was met with appraisal in parts of the organization, others were reluctant. Increased team autonomy made many of the coordinating roles in the old organization superfluous, leaving people frightened and insecure.

The theorizing in this paper is based on a single case study – with the limitations this implies. For example, results based on single case studies are not necessarily generalizable across cases. In addition, our investigation of platformization was performed in a public organization in a Scandinavian country and might not be applicable to other contexts.

Another limitation is that the transformation from silo-organization to platform organization is still ongoing. Although the transformation is promising in terms of increased generativity, we cannot be certain that the outcome will be successful.

References

Aanestad, M., and Jensen, T. B. 2011. "Building Nation-Wide Information Infrastructures in Healthcare through Modular Implementation Strategies," *The Journal of Strategic Information Systems* (20:2), pp. 161-176.

Baldwin, C. Y., and Woodard, C. J. 2009. "The Architecture of Platforms: A Unified View," *Platforms, markets and innovation*), pp. 19-44.

Boudreau, K. 2010. "Open Platform Strategies and Innovation: Granting Access Vs. Devolving Control," *Management Science* (56:10), pp. 1849-1872.

Boudreau, K. J. 2012. "Let a Thousand Flowers Bloom? An Early Look at Large Numbers of Software App Developers and Patterns of Innovation," *Organization Science* (23:5), pp. 1409-1427.

Brown, J. S., and Duguid, P. 2001. "Knowledge and Organization: A Social-Practice Perspective," *Organization science* (12:2), pp. 198-213.

Bygstad, B. 2015. "The Coming of Lightweight It," *ECIS*.

Bygstad, B., and Hanseth, O. 2018. "Transforming Digital Infrastructures through Platformization,").

Ciborra, C. 2000. *From Control to Drift: The Dynamics of Corporate Information Infastructures*. Oxford University Press on Demand.

Constantinides, P., Henfridsson, O., and Parker, G. G. 2018. "Introduction—Platforms and Infrastructures in the Digital Age." INFORMS.

de Reuver, M., Sørensen, C., and Basole, R. C. 2017. "The Digital Platform: A Research Agenda," *Journal of Information Technology*).

Dingsøyr, T., Nerur, S., Balijepally, V., and Moe, N. B. 2012. "A Decade of Agile Methodologies: Towards Explaining Agile Software Development," *Journal of Systems and Software* (85:6), pp. 1213-1221.

Dyba, T. 2000. "Improvisation in Small Software Organizations," *IEEE Software* (17:5), pp. 82-87.

Eaton, B., Elaluf-Calderwood, S., Sorensen, C., and Yoo, Y. 2015. "Distributed Tuning of Boundary Resources: The Case of Apple's Ios Service System," *MIS Quarterly: Management Information Systems* (39:1), pp. 217-243.

Eisenhardt, K. M. 1989. "Building Theories from Case Study Research," *Academy of management review* (14:4), pp. 532-550.

Feathers, M. C. 2005. *Working Effectively with Legacy Code*. Safari Tech Books Online.

Fitzgerald, B., and Stol, K.-J. 2017. "Continuous Software Engineering: A Roadmap and Agenda," *Journal of Systems and Software* (123), pp. 176-189.

Gawer, A., and Cusumano, M. A. 2014. "Industry Platforms and Ecosystem Innovation," *Journal of Product Innovation Management* (31:3), pp. 417-433.

Ghazawneh, A., and Henfridsson, O. 2013. "Balancing Platform Control and External Contribution in Third-Party Development: The Boundary Resources Model," *Information Systems Journal* (23:2), pp. 173-192.

Grisot, M., Hanseth, O., and Thorseng, A. A. 2014. "Innovation of, in, on Infrastructures: Articulating the Role of Architecture in Information Infrastructure Evolution," *Journal of the Association for Information Systems* (15:4), p. 197.

Gruhn, V., and Schäfer, C. 2015. "Bizdevops: Because Devops Is Not the End of the Story," Cham: Springer International Publishing, pp. 388-398.

Hanseth, O. 2001. "Gateways—Just as Important as Standards: How the Internet Won the "Religious War" over Standards in Scandinavia," *Knowledge, technology & policy* (14:3), pp. 71-89.

Hanseth, O., and Lyytinen, K. 2010. "Design Theory for Dynamic Complexity in Information Infrastructures: The Case of Building Internet," *Journal of Information Technology* (25:1), pp. 1-19.

Helmond, A. 2015. "The Platformization of the Web: Making Web Data Platform Ready," *Social Media+ Society* (1:2), p. 2056305115603080.

Henfridsson, O., and Bygstad, B. 2013. "The Generative Mechanisms of Digital Infrastructure Evolution," *MIS quarterly* (37:3).

Henfridsson, O., Nandhakumar, J., Scarbrough, H., and Panourgias, N. 2018. "Recombination in the Open-Ended Value Landscape of Digital Innovation," *Information and Organization* (28:2), pp. 89-100.

Hepsø, V., Monteiro, E., and Rolland, K. H. 2009. "Ecologies of E-Infrastructures," *Journal of the Association for Information Systems* (10:5), p. 430.

Humble, J., and Molesky, J. 2011. "Why Enterprises Must Adopt Devops to Enable Continuous Delivery," *Cutter IT Journal* (24:8), p. 6.

Islind, A. S., Lindroth, T., Snis, U. L., and Sørensen, C. 2016. "Co-Creation and Fine-Tuning of Boundary Resources in Small-Scale Platformization," Cham: Springer International Publishing, pp. 149-162.

Kallinikos, J., Aaltonen, A., and Marton, A. 2013. "The Ambivalent Ontology of Digital Artifacts," *Mis Quarterly*), pp. 357-370.

Langley, A. 1999. "Strategies for Theorizing from Process Data," *Academy of Management*).

Larman, C., and Basili, V. R. 2003. "Iterative and Incremental Developments. A Brief History," *Computer* (36:6), pp. 47-56.

Lusch, R. F., and Nambisan, S. 2015. "Service Innovation: A Service-Dominant Logic Perspective," *MIS quarterly* (39:1).

Monteiro, E., Pollock, N., and Williams, R. 2013. "Innovation in Information Infrastructures: Introduction to the Special Issue," *Conference on Information Systems*, p. 8.

Myers, M. D. 2013. *Qualitative Research in Business and Management*. Sage.

Pan, S. L., and Tan, B. 2011. "Demystifying Case Research: A Structured–Pragmatic–Situational (Sps) Approach to Conducting Case Studies," *Information and Organization* (21:3), pp. 161-176.

Parker, G. G., Van Alstyne, M. W., and Choudary, S. P. 2016. *Platform Revolution: How Networked Markets Are Transforming the Economyand How to Make Them Work for You*. WW Norton & Company.

Pipek, V., and Wulf, V. 2009. "Infrastructuring: Toward an Integrated Perspective on the Design and Use of Information Technology," *Journal of the Association for Information Systems* (10:5), p. 447.

Rolland, M., Rai. 2018. "Organizational Enactment of Digital Platforms:

the Interactions between Digital Options and Digital Debt," *Information Systems Research*).

Svahn, F., Mathiassen, L., and Lindgren, R. 2017. "Embracing Digital Innovation in Incumbent Firms: How Volvo Cars Managed Competing Concerns," *MIS Quarterly* (41:1).

Tilson, D., Lyytinen, K., and Sorensen, C. 2010a. "Desperately Seeking the Infrastructure in Is Research: Conceptualization of" Digital Convergence" as Co-Evolution of Social and Technical Infrastructures," *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*: IEEE, pp. 1-10.

Tilson, D., Lyytinen, K., and Sørensen, C. 2010b. "Research Commentary—Digital Infrastructures: The Missing Is Research Agenda," *Information systems research* (21:4), pp. 748-759.

Tiwana, A. 2014. "Separating Signal from Noise: Evaluating Emerging Technologies," *MIS Quarterly Executive* (13:1).

Wareham, J., Fox, P. B., and Cano Giner, J. L. 2014. "Technology Ecosystem Governance," *Organization Science* (25:4), pp. 1195-1215.

Yin, R. K. 2009. "Case Study Research: Design and Methods. Essential Guide to Qualitative Methods in Organizational Research (Vol. 5)," *The Information Systems Research Challenge (Harvard Business School Research Colloquium). London: Sage*.

Yoo, Y., Boland Jr, R. J., Lyytinen, K., and Majchrzak, A. 2012. "Organizing for Innovation in the Digitized World," *Organization science* (23:5), pp. 1398-1408.

Yoo, Y., Henfridsson, O., and Lyytinen, K. 2010. "Research Commentary—the New Organizing Logic of Digital Innovation: An Agenda for Information Systems Research," *Information systems research* (21:4), pp. 724-735.

Zittrain, J. 2008. *The Future of the Internet--and How to Stop It*. Yale University Press.

Zittrain, J. L. 2006. "The Generative Internet," *Harvard Law Review*), pp. 1974-2040.