6-18-2022

Shifting Complexity: The Impact of Modularization on IS Complexity

Erik Jochemsen
Vrije Universiteit Amsterdam, ejochemsen@gmail.com

Bart van den Hooff
Vrije Universiteit Amsterdam, b.j.vanden.hooff@vu.nl

Marijn Plomp
Vrije Universiteit Amsterdam, m.g.a.plomp@vu.nl

Mohammad Rezazade Mehrizi
Vrije Universiteit Amsterdam, m.rezazademehrizi@vu.nl

Koen Mol
Vrije Universiteit Amsterdam, koen-mol@hotmail.com

SHIFTING COMPLEXITY: THE IMPACT OF MODULARIZATION ON IS COMPLEXITY

Research Paper

Erik Jochemsen, Vrije Universiteit Amsterdam, The Netherlands, e.j.jochemsen@vu.nl

Bart van den Hooff, Vrije Universiteit Amsterdam, The Netherlands, b.j.vanden.hooff@vu.nl

Marijn Plomp,Vrije Universiteit Amsterdam, The Netherlands, m.g.a.plomp@vu.nl

Mohammad Rezazade Mehrizi, Vrije Universiteit Amsterdam, The Netherlands, m.rezazademehrizi@vu.nl

Koen Mol, Vrije Universiteit Amsterdam, The Netherlands, koen-mol@hotmail.com

Abstract

Many organizations strive to reduce the complexity of their IS architecture. One frequently used intervention is to minimize IS complexity by applying modularization. With modularization, systems are decomposed into modules to maximize IT flexibility. However, very few organizations achieve the anticipated benefits when introducing modularization. Therefore, our aim in this paper is to increase our understanding of the effects of modularization on IS complexity. Through a case study, we explore the relation between modularization and IS complexity. The results show three design choices through which modularization affects IS complexity: (1) the degree of granularity, (2) unification of terminology, and (3) the centralization of connections. The results also show that IS complexity is often shifted instead of reduced. Complexity seems to shift as a 'waterbed effect' across the IS architecture and across the different conceptualizations of complexity. With that understanding, IS complexity can be moved to a location in an organization better suited to deal with it.

Keywords: Modularization, Service Oriented Architecture, IS Complexity

1. Introduction

Organizational agility is increasingly crucial for companies to be able to respond to their customers' requirements in terms of faster service delivery, dynamically changing products, more flexibility and ease of use. This, in turn, leads to an increasing need for flexibility and agility in adjusting the technical and functional possibilities of the Information Systems architecture of organizations (Mikalef, Pateli and Van de Wetering, 2020). Rapidly changing business functionalities need to be supported by a variety of digital technologies – technologies which in themselves are also subject to rapid changes. These developments lead to increasingly complex IS architectures. Over time, these architectures have grown into complicated patchworks of soft- and hardware solutions (both legacy and brand new), from a mix of vendors or built in-house, characterized by a tight coupling between applications and processes (Benbya, Nan, Tanriverdi and Yoo, 2020; Haki, Beese, Aier and Winter, 2020; Mocker and Boochever, 2020; Schmidt and Buxmann, 2011; Widjaja and Gregory, 2020). This complexity not only limits the flexibility to introduce new products

and services, or adapt existing ones, it also makes managing and maintaining the IS architecture more difficult. A complex, tightly coupled architecture is a barrier to organizational agility (Van Oosterhout, Waarts and Van Hilligersberg, 2006), as changing information systems in response to fast changing business needs becomes very complicated: a tightly coupled architecture is typically very rigid, and maintenance or changes are often costly (Bhatt et al., 2010; Widjaja et al., 2012).

One often used method to reduce the complexity of IS architectures, and thereby enhance organizational agility, is implementing a modular architecture (Bhatt, Emdad, Roberts and Grover, 2010; Mikalef et al., 2020; Papazoglou and Van Den Heuvel, 2007). Modularization means pursuing loose coupling through implementing a common standardized backbone and developing or acquiring modular pieces of software that can be easily combined and re-used (Markus, Tanis and Van Fenema, 2000). Modularization initiatives should reduce IS complexity by "encapsulation and minimizing the requirements for shared understanding by defining service interfaces in an unambiguous and transparent manner" (Papazoglou and Van Den Heuvel, 2007, p. 391). In other words, the ease of combining and reusing modules should make it easier to adapt individual components of a system without compromising the entire system itself. These architectural concepts mainly help in addressing IS complexity by breaking the systems landscape down into independent modules to increase flexibility and better handling. Modules are designed independently of one another, but must function together as a whole (Baldwin and Clark, 2006; Miller and Elgard, 1998).

While many organizations introduce forms of modularization (such as Service Oriented Architectures), they often fail to realize anticipated benefits, such as increased IT flexibility or organizational agility (Joachim, Beimborn and Weitzel, 2013). Joachim et al. (2013) contend that this failure is often not so much due to the technical characteristics of such architectures, but rather to governance decisions. We want to understand the paradoxical impacts of modularization on IS complexity and why it sometimes reduces and sometimes decreases IS complexity. Beyond that, current IS research provides only limited insight into how the complexity of IS architectures is affected by modularization. We want to question the assumption that there is only one way of understanding modularization. One of the reasons for this gap is the fact that there are a variety of conceptualizations of IS complexity, which are often used interchangeably (Widjaja and Gregory, 2020), leading to different conclusions regarding the effects of different interventions. In order to provide more insight into this matter, this paper answers the question: *How does modularization impact the complexity of IS architectures?*

To understand what the relationship is between modularization and IS complexity, extant research on IS complexity and modularization is reviewed. Next, we empirically explore the impact of modularization on IS complexity, by reporting on a case study conducted at a European Bank. The impacts of modularization on IS complexity provide an outline for answering the main research question. This contributes to a better understanding of the effects of modularization on IS complexity. By researching the different ways of implementing service orientation on the different conceptualizations of IS complexity, we obtain a deeper understanding of how modularization can have different impacts for different conceptualizations of IS complexity. We show that modularization not simply reduces complexity, but leads to shifts in IS complexity. Organizations that introduce service orientation can optimize the effects of modularization by applying the implementation choices in certain ways to shift complexity towards a place where it can be managed more effectively.

2. Theory

In this section we discuss the existing literature on modularization and IS complexity. We examine different conceptualizations of IS complexity and how these relate to organizations' efforts to introduce modular architectures.

2.1 IS complexity

In this paper, we focus on the complexity of *IS architectures*, which refers to "the overarching structure and properties of the relationships among the systems and applications in an organization's IT portfolio" (Tiwana and Konsynski, 2010, p. 288). As argued before, this complexity is rapidly increasing, as legacy systems are combined with more and more emerging technologies like cloud computing, big data and analytics, internet of things, and blockchain (Ross et al., 2019). The complexity of an organization's IS architecture is often conceptualized in terms of "the quantity and variety of system elements and the relationships between these" (Schmidt, 2015, p. 244). In terms of Schneider et al.'s (2014) multidimensional framework of IS complexity, this can be characterized as a *structural* perspective on complexity, as it captures the complexity "inherent in a static snapshot of a system" (Beese et al., 2016, p. 5) – what the architecture looks like at a certain point in time. Although the structural conception of IS complexity is quite often used to indicate the complexity of IS architectures, the literature shows that complexity is more than this snapshot of what a system looks like (Schneider, Zec and Matthes, 2014; Widjaja and Gregory, 2020).

First, there is also a *dynamic* conception of IS complexity. The dynamic complexity of an IS architecture arises from the interactions over time, between the different technological components of which this architecture consists, which dynamically and nonlinearly influence one another and may lead to unpredictable outcomes (Haki et al., 2020). Dynamic complexity is particularly relevant and critical as both the business and IT environments of organizations are constantly changing (Furstenau et al., 2019; Xia and Lee, 2005). Dynamic complexity manifests itself when an IS architecture exhibits behaviors that are not easily reducible to the properties of its components and relations (Haki et al., 2020; Hassan, 2014). As Burns et al. (2012) state, the behavior of these systems is nonlinear and exceptionally unpredictable. Hassan (2014) refers to these behaviors as "emergent properties" of a system. Since a complex system contains many elements and interactions that continually change in different ways, predicting behavior is very difficult. In the dynamic conception of IS complexity, a system can be very complex even if it has a relatively small number of components and interactions – when it shows emergent properties or behaviors that cannot be easily comprehended, it is still a complex system (Benbya et al., 2020; Haki et al., 2020; Widjaja and Gregory, 2020).

Finally, we can also distinguish a *subjective* conception of IS complexity (Jochemsen et al., 2016; Widjaja and Gregory, 2020). Here, it is not so much the technology itself, but the interactions of users with the technology that defines the complexity of a system – complexity is 'in the eye of the beholder'. Different users and user groups have different technological frames (assumptions, knowledge, and expectations in regard to technology) towards the systems they use (Orlikowski and Gash,1994). Understanding how different stakeholder groups view complexity can greatly help with organizational issues during the implementation of any system (Mocker and Boocheever, 2020).

2.2 Modularization

To manage the increased complexity of their IS architectures, and to improve their flexibility and agility, organizations apply the concept of modularity in their IS architecture (Joachim et al., 2013; Mikalef et al., 2020; Tiwana and Konsynski, 2010). Modularity is defined by Schilling (2000, p. 312) as "a continuum describing the degree to which a system's components can be separated and recombined, and it refers to both the tightness of the coupling between components and the degree to which the rules of the system architecture enable the mixing and matching of components". A modular system is a loosely coupled system, which can be decomposed into multiple modules that each can operate individually and separately from each other (Baldwin and Clark, 2006). The individual modules can then be combined into new products or services. This maximizes the flexibility of IS by reducing overlapping functionalities, and isolates complexities in the specific modules (Mikalef et al., 2020).

Modularity can be designed in terms of the granularity of modules, the mapping of design elements to the modules, the interactions among the design elements within each module, and interfaces or interactions between modules (Ethiraj and Levinthal, 2004; Miller and Elgard, 1998). For each module, it needs to be defined how much functionality is built into a module. Defining an appropriate design for the modules within the IS architecture is crucial in balancing reusability of the modules, the duplication of functionality, and the number of modules (Dörbecker et al., 2002). The connectivity and compatibility of different modules within an IS architecture define the required interchangeability and combination of functionalities. This requires that the modules have standardized interfaces and interactions and that each module is independent (Miller and Elgard, 1998). This also defines the level of scalability and reusability of modules.

One common form of implementing loose coupling and modularization within an IS architecture is through a Service Oriented Architecture (SOA). In a SOA, application software is divided into separate services, where a service is "an independent block of software code that can be naturally reused and composed with other services to create a system" (Teixeira et al., 2015, p. 5366). Bieberstein et al. (2005, p. 5) define a SOA as: "a framework for integrating business processes and supporting IT infrastructure as secure, standardized components – services – that can be reused and combined to address changing business priorities". In a SOA, a high degree of granularity refers to many services, where any one service has limited functionality. A higher level of granularity helps reduce duplication in functionality and functional dependencies between components. Service orientation standardizes the integration patterns between services (modules) and replaces point-to-point interfaces with standardized interfaces between services and an Enterprise Service Bus (ESB) as a centralized platform (Baldwin and Woolard, 2009; Schuetz et al., 2013; Widjaja et al., 2012). As SOA facilitates easier handling of increasing numbers of users, workload, or transaction volumes (Chanopas et al., 2006), it seems that implementing service orientation positively influences scalability of an IS architecture. As Joachim et al. (2013) argue, SOA is generally expected to exhibit higher scalability than point-to-point connections, because an ESB is applied for application integration.

2.3 IS complexity and the impact of modularization

In this section, we describe the relation between modularization and IS complexity for each of the three conceptualizations of IS complexity.

2.3.1 Modularization and structural complexity

The literature on complex systems argues that modular architectures are particularly useful in managing the complexity of systems (Ethiraj and Levinthal, 2004). This specifically applies to a structural conceptualization of complexity, in terms of the quantity and variety of system elements and the relationships between these elements. The standardization inherent to a modular approach tends to reduce this variety, thus making the complexity of a modular system more manageable than an integrated system (Ethiraj and Levinthal, 2004). Granularity is an important choice here, as this determines the size and number of modules, as well as the extent to which the complexities of a system are contained within separate modules and are separated and insulated from other modules. The complexities can also be reduced by the way the modules are integrated. This can be illustrated through a simple network structure with front-end and back-end components, where all front-end components are connected with all back-end components. In this situation the number of integrations is defined by the number of front-end components (FE) times the number of back-end components (BE) (so FE * BE). When considering a system with an Enterprise Service Bus (ESB) that routes the integration between the modules, the number of links is FE + BE. This means that the number of links between components scales much more slowly as more components are added in a system with an ESB. This also relates to the scalability of the system and is explained by Papazoglou and Van den Heuvel (2007, p. 393): "point-to-point integration is not scalable and very complex as the number

of integration points increases as the number of systems increases and can quickly become unmanageable. Hence, the broad use of ESB supporting a variety of hub-and-spoke integration patterns".

Furthermore, scalability is impacted by the need for additional applications or services. The effort to scale up the system is relatively small, because of the possibility of reusing existing components (Chanopas et al., 2006). This means that with a modular approach, the number of applications, measured in terms of number of components and interactions, increases at a slower rate than in regular IS architectures that do not reuse functionalities (Benaroch, 2013; Schneider et al., 2014). Because of the reusability of services in SOA, fewer applications are needed in a firm. Less applications typically also means less heterogeneity between the components. SOA helps decrease the heterogeneity of connections in a system and reduces the IS complexity from that perspective.

2.3.2 Modularization and dynamic complexity

In terms of dynamic complexity, a system is considered complex when it shows emergent properties or behaviors that cannot be easily reduced to individual components. Modularization splits a complex system into modules, multiplying the design options. This increase (or decrease) in granularity can impact the complexity of the system as a whole. The adoption of modularity in complex systems can generate emergent behavior (Baldwin and Clark, 2006). The modules need to integrate and work together (Tassey, 1999). This moves decisions from a central point of control within the system towards the individual modules, allowing for the decentralized system to evolve in new ways (Baldwin and Clark, 2004). Because of the emergent nature of these properties, it is difficult to predict the outcomes in terms of the level of complexity of a system (Hassan, 2014). Determining the impact of modularization on the dynamic complexity of a system would require a further examination of how behavioral patterns in the system are affected, rather than its components and relations.

2.3.3 Modularization and subjective complexity

Prior to modularization, users perceive the functionality of an application that spans the entire system. After modularization, users see what they previously saw as 'the system' being fragmented, because the functionality of the application is split into individual components and the system as a whole is no longer recognizable any longer as a single entity (Shapiro and Varian, 1999). As subjective complexity is entirely in the eye of the beholder(s), the level of complexity of a system decreases when agents learn about a system. The reduced visibility of some of the modules and the integration of the different modules (likely via an ESB) could lead to a shift in how specific groups view the complexity of an IS architecture. Joachim et al. (2013) remark that there is a "dearth of research on SOA governance and its mechanisms" and that many organizations struggle with implementation, not because of technical difficulties, but because of managerial difficulties in SOA implementation in similar ways as it is to discontinue established systems (Mehrizi et al., 2019). The SOA governance mechanisms developed by Joachim et al. (2013) and Kohnke et al. (2008) help bridge the gap between different technological frames, and streamline the learning process and decision making through actively influencing these frames of different groups in an organization.

In conclusion, the literature indicates that modularization involves different choices, and also that IS complexity has multiple dimensions. However, extant research provides limited understanding of how various modularization choices impact different complexity dimensions. Based on our overview of the literature, there does not seem to be a straightforward relation between modularization and complexity. Therefore, we further explore the relationships between these concepts in an exploratory case study.

3 Method

In this section, we discuss the methodological details of the empirical study. To enhance our understanding of the effects of modularization on IS complexity, we conducted a case study on a SOA implementation. The research methods and design are further explained below.

3.1 Research design

As discussed in section 2, the current literature provides limited insight into the relationship between modularization and the different conceptualizations of IS complexity. In order to gain insight into these relationships, several steps were undertaken. First, a case was selected in which rich information was available about both modularization (i.c. SOA) and its effects on IS complexity. A single case study is appropriate since the research is exploratory (Eisenhardt and Graebner, 2007). It helps to build theories by enabling pattern identification of rich observations from the field (Saunders et al., 2015). Second, introductory meetings were held with key stakeholders to gain insight into the previous and current situation regarding the SOA implementation. Third, interviews were held with different stakeholders throughout the implementation process. Fourth, the interviews were analyzed in order to gain further insights and identify patterns in the interaction between SOA and different conceptualizations of IS complexity. Furthermore, the first author was directly involved in the design and decision making process of the program. In his role as IT strategy consultant and facilitator during that process, he obtained an in-depth insight in initial problem statements, alternative solutioning, decision documents, and design documents. This allowed us to get more insight in the context and background of the observations made in the interviews. During discussions with the other authors we ensured we avoided any bias of observation during the case study.

3.2 Research context

The case study was conducted at Alpha Bank (anonymized). Alpha Bank is a mid-sized Western European bank. Alpha Bank's strategic goal is reducing IS complexity by implementing SOA. This large-scale attempt at implementing SOA provides a rich setting to study its effects on IS complexity. The initial reason for implementing SOA was the large number of legacy systems Alpha Bank used, sometimes dating back to the 1960s. These legacy systems were built as added functionality over the years, or acquired in one of various mergers and takeovers. These legacy systems are connected with point-to-point interfaces. Many interviewees used the metaphor 'spaghetti' when talking about Alpha Bank's IT architecture, referring to the large number of (hidden) connections and interdependencies between systems in their IT architecture. The result is that employees do not exactly know how systems work anymore. For example, one interviewee noted, referring to having to rebuild old functionality, that: "*You don't know what you will find. It is also very difficult to find out what the function of such a system is. Everybody understands the basic function, but beyond that... To understand that, someone has to technically crawl through the entire system to find out that somewhere something is off.*" This presence of many systems, that have many connections, that are not standardized, points to a high degree of complexity before the implementation of SOA. Next to reducing numerous interdependencies between IT systems, Alpha Bank aims to reduce the number of redundant duplicate functionalities that evolved over the years. In their approach, this is denoted by trying to make 'reusable' services, aiming to become more agile with less IT costs and a faster time-to-market.

3.3 Data collection and analysis

This research was conducted within several departments of Alpha Bank. The main data collection method was the semi-structured interview. Before these interviews, two introductory meetings were held, in the form of open-ended conversations with employees involved in the (early stages of the) SOA implementation. These conversations were largely meant to prepare for the interviews. The high degree of freedom in these open-ended conversations allowed for rich and full stories, providing an overview of Alpha

Bank's timeline and organization with regard to implementing SOA. In total, 24 interviews were conducted, 12 specifically on the implementation of service orientation and 12 interviews covered the topic of the implementation of SOA as part of a larger investigation into IS complexity within Alpha Bank. We conducted semi-structured interviews with three groups of stakeholders. The first group consists of informants from departments in different stages of building services as a service provider. The second group of informants are from the central department managing the ESB and responsible for connecting new services. The third group are the consumers of the services, both within the IT department and consumers within the Retail & Private Banking and Corporate Banking business units. All three groups consisted of both managers and engineers. In addition, we used internal documents from the SOA implementation program that described scope and objective and reports on the implementation approach. These secondary data sources helped in interpreting statements that were made in the interviews and provided context and clarification.

The empirical data was first coded based on the three conceptualizations of IS complexity as described in section 2.3: (1) Structural Complexity, (2) Dynamic Complexity, and (3) Subjective Complexity to understand how each aspect of IS complexity emerged in the case We analyzed the data by examining how SOA was applied and its consequences on the three dimensions of IS complexity. In this process of coding, three design choices emerged through which Services Oriented Architecture influences IS complexity: degree of (1) granularity, (2) unification of terminology, and (3) centralization of connections.

4 Findings

In this section we discuss how three choices have impacted the different types of complexity. Table 1 shows the consequences for complexity listed as a higher degree of each of the design choices.

4.1 Granularity

Our findings show that the level of granularity of modules influences the degree of *structural complexity*. At first sight, a low level of granularity – a lower number of services, each with a broad range of functionalities – seems to reduce structural complexity, because this means a lower number of components and interactions. But in this situation, much of the complexity is hidden *inside* the service, as one service contains a broader range of functionality. The result is that individual services become more difficult to manage and replace. To illustrate this effect, one interviewee described a case in which a single archiving service was built consisting of 31 operations:: "*Every time we have to update one of these 31 components, we have to bring a new version of the service online. We now have version 1.0 and version 1.1, because users asked for additional functionality. Now we are getting requests for new data structures, so we are building version 2.0.*". Furthermore, a lower level of granularity led to large services that had to be changed for different business users. This led to duplicates with different versions of one service, increasing the number of components in an IS landscape and thus increasing structural complexity. Regarding *dynamic complexity,* we found examples of unexpected behaviors caused by the dynamic interaction between humans and technological aspects. One example was given by the following interviewee: "*Many people don't know exactly how it [IS] works, which is troublesome. What you get is unexpected behavior from the services. So you think you know how it works, you think it will be alright. However, different [technical] activities are taking place inside and between the services.*" So although interviewees indicated that the interactions between technical and human actors are difficult to predict, we did not see a clear relationship between the level of granularity and dynamic complexity.

	Granularity	**Unification of terminology**	**Centralization of connections**
Structural complexity	● Complexity decreases within the modules as with smaller modules there are less functionalities incorporated within each of the modules. ● Complexity increases between modules as a result of having more components for services and connections across the IS architecture.	● Complexity decreases with lower number of services required because of unified terminology (that can be reused).	● Complexity decreases as a result of lower number of connections between services by using a central gateway in between.
Dynamic complexity	● Complexity decreases within the smaller service modules that contain less functionalities. ● Complexity increases with more services between services.	● Complexity increases because single terminology covers a wide range of definitions and meanings with possible misinterpretation or overgeneralizations.	● No noticeable observations that relate to the level of centralization of connections.
Subjective complexity	● Complexity decreases for providers as a result of increased manageability of smaller individual services. ● Complexity increases for consumers as a result of a higher number of smaller services that they have to assemble to form their required business functionality.	● Complexity decreases for providers as a result of lower heterogeneity of connections between services. ● Complexity increases for consumers as a result of higher internal complexity of connections between services.	● Complexity decreases for providers as result of less dependencies on different consumers to upgrade or modify services. ● Complexity increases for consumers as a result of higher dependency on a central ESB team and reduced feeling of autonomy.

Table 1. Impact of implementing SOA on the conceptualizations of IS complexity

The degree of granularity was also found to have an effect on *subjective complexity*. Here we see differences between providers and consumers in how they perceive complexity. A low level of granularity is perceived as less complex by consumers, because they have to deal with fewer services, and they are not confronted with the internal complexity of these services. Conversely, a high degree of granularity was perceived as being more complex by consumers – when there are many services, consumers find it difficult to know which ones to use and how to combine them to perform a simple task (e.g., archiving a document in this situation might need four services: make dossier, put document in dossier, change metadata, close dossier). So they prefer larger, more 'business meaningful' services. This was described as follows by one of the consumers: "*I don't want different services for 'make account', 'make package' and 'make card' I just want 'make customer'.*" Providers, on the other hand, generally perceive a higher degree of complexity when granularity is lower, because they are confronted with the services' internal complexity, whereas they perceive a high degree of granularity as less complex. This was due to the increased manageability of (individual) smaller components. This led to many discussions on how many data attributes should be included in each of the services.

4.2 Unification of terminology

To achieve the desired service architecture, Alpha Bank chose to unify the terminology being used across all services. However, the initial attempt in implementing this radically new model failed due to organizational resistance. One of the reasons was to, as one interviewee put it: "*put an end to people talking about different concepts*." A simple example of this was given by somebody from the SOA center of expertise: "*Say we are talking about a customer, what is a customer? Is that an account-holder? Or a customer in the technical process?*" The unification of terminology aimed to ensure optimal consolidation of both technical and non-technical elements.

Unification of terminology could have reduced *structural complexity*, because it standardizes the language between services. The unification of terminology did allow for re-use of services. At the same time, however, like with the degree of granularity, moving towards a more unified model ment that the (structural) IS complexity was hidden inside more complex data definitions of each of the services. Individual definitions became more complicated, as they had to be more generically applicable throughout the bank. This is because generic definitions had to be suited to be applied for many different functionalities. One interviewee on the provider side described this as follows: "*You are actually trying to describe all functionality in a bank with a single unified language. For example for a bank account, that means the account is always the same, irrespective of if you are talking to domain X, Y, or Z. But in reality an account for domain Z is something completely different than an account for domain X... So we should be putting everybody who thinks Z in one model. But now they are unifying X and Z, which makes an account incredibly complex. If you look at an account after the unification, you will think: what a monster!*" This means that while the structural complexity (in terms of the heterogeneity of links) was reduced by the unification of terminology, individual definitions became elaborate and more complex. The current situation at Alpha Bank with regard to terminology, however, is not very unified. This means that many different definitions are used for any term, leading to a high degree of structural complexity because of a high level of heterogeneity of services.

Looking at the *dynamic complexity*, one interviewee from the SOA Center of Expertise described the situation of decentralized terminology as follows: "*Everything uses its own data definitions. In one system something is called a beginning date and in another system it is called a start date. On a higher level, that makes it difficult to understand how a system performs a certain task, which steps it performs. Because they [services] are so intertwined and because they don't work in a universal way, everything works in its own way. That makes the behavior of systems unpredictable. The risks are that you don't understand what the consequences are of a small change in a system for the performance of the total picture.*" An example was given by another interviewee from the central ESB team, who said: "*That knowledge is not in our domain, we need the back-end domains for that. To get a clear image, we need to ask ourselves how such a transaction works. That's hard because the knowledge is hard to get from the colleagues of those systems. Many of them also don't know exactly how it works.*" So, a decrease in structural complexity (heterogeneity of links) caused an increase in unexpected behavior because services contained more abstraction and different functional applications of a single term at the same time within the same service. The application of a single term for multiple functional applications added more complexity within each of the services, because it needed to comply with different business logics at the same time and needs to continue doing so after each modification or change within that service's operations.

As for *subjective complexity*, our findings show that consumers and providers perceived varying degrees of complexity depending on how unified the terminology is. On the one hand, consumers, who always had their own independent definitions, experienced more complexity. For some consumers a more unified terminology added unnecessary aspects to a definition, making working with it a more complex experience. Furthermore, consumers and providers from all over the organization needed to agree on definitions and changes in these definitions. On the other hand, for service providers, the complexity stemmed from having

a multitude of different terminologies that were used by different consumers, but managed by a single provider. As one person on the provider side put it: "*it is very easy to misinterpret the definitions in work from other departments.*" The technical aspects of unifying terminology, while ensuring lower heterogeneity of services, become very challenging. The internal functionality within a service increased the perceived complexity. Higher unification led to an increase of perceived complexity on the consumer side, as the links caused an increase in unexpected behavior that they had difficulty with to interpret. Low unification caused unexpected dynamics in services, as the increase of heterogeneity of services caused an increase in subjective complexity for providers.

4.3 Centralization of connections

The third design choice is about the centralization of connections that refers to the degree to which a central ESB was used as a central gateway. At Alpha Bank we saw that the *structural complexity* decreased with a higher degree of centralization of connections, since this led to a reduction of the number of interfaces between service operations. This effect was described as: "*The service component model... you can see that people have thought about the number of links. So you get less links, and the landscape becomes less complex and more manageable.*" We saw less components and interfaces, but more connections that all fed into a single gateway and an extra layer (and component) in the IS architecture.

Regarding *dynamic complexity*, we did not make any noticeable observations that relate to the level of centralization of connections. One explanation could be that the interviews were held in the initial build-up phase of the SOA at Alpha Bank. Therefore, we observed more references to aspects that are related to the (organizational) build-up phase and less yet that are related to (operationally) running a SOA.

For providers, having a central technical gateway decreased their *subjective complexity*. Most providers felt their autonomy increased and cited less dependencies on consumers, as they only had to worry about the single connection to the ESB when changing their systems: "*If you have a number of point-to-point interfaces and you want to change something, you have to discuss the change with all different stakeholders.*" Where most providers felt their autonomy increased, consumers generally felt their autonomy had decreased, which meant increased complexity in how they dealt with IT. Because all technical components now had a dependency on the ESB, the ESB team typically needed to be involved in building or connecting services. Consumers feel that the expertise they have could not be translated by the ESB team: "*Everything used to be done in autonomous environments. If services have to go through the ESB, departments need help. Having the service built by others decreases autonomy.*" Another consumer added: "*We had an alternative. We did have a number of services that were available. They did have old technology and were not consistent. But, the people within the customer channels domain knew how to use them. So it is difficult to convince people to use the new services.*" In some cases this resulted in workarounds. One interviewee from the SOA center of expertise described this as follows: "*Because we chose to do this centrally, we have created dependencies in the organization. The organization doesn't like that. If Channels need something from Payments, they are tempted to go to them directly and not to involve the ESB team.*" The interviews indicate that the dependency on the ESB team was largest in the phase of building a service; once the service was built, the dependency tended to decrease.

4.4 Shifting complexity

Our findings show that the modularization impacted IS complexity via design choices made in terms of the degree of granularity, centralization of terminology, and centralization of connections. We did not find, however, a simple relationship where an increase in centralization of connections led to a decrease in complexity, but rather an increase in one area ánd a decrease in other areas át the same time. Therefore, the picture that emerged from our findings is not that modularization led to a *reduction* of complexity –

rather, we see a *shifting* of complexity. We observed this shifting of complexities along two main dimensions (1) from *within* applications to *between* services, and (2) between *providers* and *consumers*.

The first is the shift of complexity *within* an application to complexity *between* service modules of the IS architecture. Complexity seems to be reduced from a *structural* conceptualization, as components (services and data definitions) themselves became less complex. At the same time, however, we saw an increase in *dynamic* complexity as more complex interactions between these components took place. As one of the engineers from the provider side mentioned: *"There are different versions of some of the services. If you connect to these and you don't understand the working of these versions you get unexpected outcomes, because one version could lead to a different outcome than another version.".* This indicates that modularization of the IS architecture of Alpha Bank moved the complexity from the global IS architecture to the local services, and diminished the dependency of the services on the larger architecture. For each of the three choices (granularity, terminology, and connections), we saw that the level of structural complexity in general decreased because of the creation of reusable services, and that the diversity of heterogeneous point-to-point interfaces was replaced by centralized connections via an ESB gateway. However, not all business functionalities could be fulfilled with one individual service, which meant that multiple services needed to be combined. This created both a functional and a technical dependency on other service(s). This dependency meant that the complexity shifted from *within* the components, to *between* the components. As the (structural) complexity decreased as a result of having less (heterogeneous) services, we saw an increase of dynamic complexity between services as a result of single terminology that covered a wide range of definitions and meanings with possible chance of misinterpretation or overgeneralizations. A lower number of services required more abstraction in these services to cover a broader functional range. But that abstraction led to functional and technical mismatches in parameters and data labels when these are combined with other services to form business functionalities. We saw an increase of complexity as a result of all connections being directed via a central ESB as a gateway that could generate more unexpected behavior and acts as a bottleneck and single point of failure.

Where the first shift of complexity was observed within the IS architecture, the second shift is one of *perceived complexity*, between providers and consumers. Depending on the choices that were made regarding granularity, unification of terminology, and centralization of connections, we saw in our case study that the perceived complexity mostly decreased for providers, but at the same time increased perceived by the consumers of services. We saw that different stakeholders with different technological frames had different perceptions of the effects of modularization. Consumers saw more complexity for adjustments in functionalities. As one consumer stated: "*For every parameter, you now have to understand what it does and what [data] will come back. Does it return a value or multiple values? If so, what are these values exactly? If I need one value but the system gives me multiple ones, which one do I need? You need to understand that kind of behavior of a transaction well to call a service correctly.*" *So,* consumers indicated an increased dependency on the central ESB team. However, for providers of services we observed that their work became simpler and more standardized. The services decoupled providers' work from the front-end side. This decoupling allowed for less dependencies on consumers for upgrades and maintenance. A statement by somebody working at the provider side: "*Technically speaking they don't need us anymore because they can just use the service on the ESB and they're done. In practice they still need help, but that is more about the function and unraveling their business process.*"

5 Discussion

Our study provides two contributions to research on IS complexity and modularity. First, we identify three design choices in the implementation of modular architectures that impact IS complexity: the degree of (1) granularity, (2) unification of terminology, and (3) centralization of connections. These three choices can

be used as dials while implementing modularization. Second, our findings show that these design choices do not have unambiguous effects on the complexity of IS architectures – in terms of reduction or increase – but rather to a shifting of complexity along two dimensions. Where we observed reduced complexity along the conceptualizations of IS complexity in certain areas, we also saw increases in other areas. Complexity seems to shift across the IS architecture and across different conceptions of complexity. In this section, we will elaborate further on these theoretical contributions, followed by their implications for practice, our study's limitations, and avenues for future research.

5.1 Design choices for modularization

The three aspects of modularization are not only a way to dissect the effects of modularization on IS complexity, but are also variables that can be further applied to the analysis of a specific implementation of IS architecture modularization. Much literature describes the effects of modularization from a point of view that it only reduces complexity. However, our study indicates that a low level of granularity results in large individual components that are hard to manage individually due to the high degree of functionality built in. This appeals to the end-to-end argument by Saltzer et al. (1984), which entails that one should include as much functionality in a module as possible to avoid dependencies between modules. Our findings show, however, that this is not unequivocally good advice: where large services are preferable (less complex) for front-end consumers, a low level of granularity is not preferable for back-end providers, who find smaller services easier to manage due to their increased modularity. As for unification of terminology, previous research indicates that this can positively influence the scalability of modules across the IS architecture (Joachim et al., 2013). But again, our findings indicate that this design choice does not have unambiguous effects: where a high level of unification of terminology reduces the heterogeneity of services (reducing structural complexity) and allows for reuse of components across the architecture, it at the same time can also impact subjective and dynamic complexity negatively, because different stakeholders use the same terminology with different meanings – which may actually *negatively* influence scalability. Finally, the centralization of connections through an ESB (or a similar standardized interface layer) reduces the number of connections in a system, decreasing structural complexity. It also centralizes processing, however, which increases the subjective complexity for consumers. For providers this increases their autonomy. An ESB as an extra layer and component in an IS architecture through which all connections are routed could also increase the complexity of the system as a whole (Schmid et al., 2021).

5.2 From reducing to shifting complexity

As a result of implementing modularization there is a shift of complexities across the IS architecture. First, we see a shift from a technological perspective of (structural) complexity from within services to between services in the IS architecture. The second shift is socio-technical, where we observe the perceived complexity shift between stakeholder groups and how consumers and providers of services perceive the complexity of the IS architecture differently as a result of implementing modularity. This idea adds to the more traditional conception of how modularization affects complexity by Papazoglou and Van den Heuvel (2007, p. 391), who state that modularization reduces IS complexity by "encapsulation and minimizing the requirements for shared understanding by defining service interfaces in an unambiguous and transparent manner". Furthermore, complexity can be reduced by increasing the modularity of a system. Modularization aims to reduce the structural complexity in terms of the number and heterogeneity of components and relations. However, our research shows that this can lead to an increase in subjective complexity and dynamic complexity. The difference between shifting complexity and reducing complexity is relevant, since it changes the manner in which IS complexity is dealt with. This means that minimizing the number of components or connections in a system for example may not necessarily be the best solution for dealing with IS complexity. Rather than minimizing such indicators, our study suggests that a balance should be found for each of the three design choices through which modularization impacts IS complexity.

These shifts in complexity provide new insights into the effects of modularization. Most existing literature describe the effect of design choices for modularization in isolation. However, the three choices for implementing modularization allow organizations to move IS complexity towards places within an organization that are better suited to deal with it. For Alpha Bank, this meant that they initially made the mistake of centralizing too much, with services with low granularity that had a fully unified terminology and very centralized responsibilities. This led to an increased subjective complexity for both providers and consumers. For providers because of the large services that require complicated unification of terminology to ensure business logic within the services, and for consumers because of the complicated unification of terminology that made it difficult to understand where the business logic resides behind the centralized ESB. Alpha Bank's reaction was to alter the choices for the SOA implementation and still centralize the connections but to unify the terminology only where duplicates across different departments would occur, and leave department specific systems, terminology, and responsibilities decentralized. They also decreased the level of granularity and made services with more functionalities in it to avoid over-fragmentation of the IS architecture.

5.3 Practical implications

Organizations often work on the assumption that complexity decreases with modularity. Organizations seem to look at the perception of the providers (and less that of consumers). Our study suggests that introducing service orientation does not necessarily reduce IS complexity; we observed a 'waterbed effect' of shifting complexities. Organizations can use the three design choices as a 'volume knob', that can be adjusted depending on the case, organizational context, and management belief. Different configuration of the three choices impacts on how IS complexity is displaced across the IS architecture.

5.4 Limitations and further research

In this paper, we analyzed how modularization influences IS complexity through a single case study. The advantage of conducting a single case study is that it enables identification of patterns (mechanisms) of rich observations from the field. It allows for the detection and rich description of the three design choices through which modularization influences IS complexity. While such a single case study is appropriate for exploratory research, it cannot be generalized. To increase the generalizability of the results, evidence should be collected across multiple organizations. By studying additional organizations, a more general model of the relation between modularization and IS complexity may appear. That generalization might also show that the observations might apply to other layers of the IS landscape (e.g. infrastructure). Lastly, future research could also help in trying to quantify the increases and decreases of the observed complexities during SOA implementations.

References

Baldwin, C. Y., and Clark, K. B. (2006). Modularity in the design of complex engineering systems. In Complex engineered systems (pp. 175-205). Springer, Berlin, Heidelberg.

Baldwin, C. Y., and Woodard, C. J. (2009). The Architecture of Platforms: A Unified View. *Platforms, Markets and Innovation*, 19-44.

Beese, J., Aier, S., Haki, K., and Aleatrati Khosroshahi, P. (2016). Drivers and Effects of Information Systems Architecture Complexity: A Mixed-Methods Study. *24th European Conference on Information Systems*. Istanbul, Turkey.

Benaroch, M. (2013). Understanding Factors Contributing to the Escalation of Software Maintenance Costs. *34th International Conference on Information Systems*, 1-15, Milan, Italy.

Benbya, H., Nan, N., Tanriverdi, H., and Yoo, Y. (2020). Complexity and Information Systems Research in the Emerging Digital World. *MISQ Quarterly*, 44(1), 1-17.

Bhatt, G., Emdad, A., Roberts, N. and Grover, V. (2010). Building and leveraging information in dynamic environments: The role of IT infrastructure flexibility as enabler of organizational responsiveness and competitive advantage. *Information and Management*, 47(7-8), 341-349.

Bieberstein, N., Bose, S., Walker, L. and Lynch, A. (2005). Impact of service-oriented architecture on enterprise systems, organizational structures, and individuals. *IBM Systems Journal*, 44(4), 691-708.

Burns, A., Nanayakkara, P., Courtney, J. and Roberts, T. (2012). Complex adaptive systems, agent-based modeling and information assurance. *18th Americas Conference on Information System*, Seattle, USA.

Chanopas, A., Krairit, D. and Ba Khang, D. (2006). Managing information technology infrastructure: a new flexibility framework. *Management Research News*, 29(10), 632-651.

Dörbecker, R., Tokar, O., and Böhmann, T. (2015). Deriving Design Principles for Improving Service Modularization Methods; Lessons Learnt From a Complex Integrated Health Care Service System. *23rd European Conference on Information Systems*, Munster, Germany.

Eisenhardt, K. M. and Graebner, M. E. (2007). Theory building from cases: Opportunities and challenges. *Academy of Management Journal*, 50(1), 25-32.

Ethiraj, S. K. and Levinthal, D. (2002). *Search for architecture in complex worlds: an evolutionary perspective on modularity and the emergence of dominant designs*. Wharton School, University of Pennsylvania.

Fürstenau, D., Baiyere, A., & Kliewer, N. (2019). A dynamic model of embeddedness in digital infrastructures. *Information Systems Research*, 30(4), 1319-1342.

Haki, K., Beese, J., Aier, S. and Winter, R. (2020). "The Evolution of Information Systems Architecture: An Agent-Based Simulation Model." *MIS Quarterly*, 44(1), 155-184.

Hassan, N. R. (2014). Systemic Complexity and Socio materiality, A Research Agenda. *20th Americas Conference on Information Systems*, Savannah, USA.

Joachim, N., Beimborn, D. and Weitzel, T. (2013). The influence of SOA governance mechanisms on IT flexibility and service reuse. *Journal of Strategic Information Systems*, 22(1), 86_101.

Kohnke, O., Scheffler, T. and Hock, C. (2008). SOA governance – Ein ansatz zum management serviceorientierter architekturen. *Wirtschaftsinformatik*, 50(5), 408-412.

Markus, M. L., Tanis, C. and Van Fenema, P. C. (2000). Enterprise resource planning: multisite ERP implementations. *Communications of the ACM*, 43(4), 42-46.

Mehrizi, M. H. R., Modol, J. R., & Nezhad, M. Z. (2019). Intensifying to cease: Unpacking the process of information systems discontinuance. *MIS Quarterly*, 43(1), 141-165.

Mikalef, P., Pateli, A. and Van de Wetering, R. (2020). IT architecture flexibility and IT governance decentralisation as drivers of IT-enabled dynamic capabilities and competitive performance: The moderating effect of the external environment. *European Journal of Information Systems*, 1-29.

Miller, T. D.and Elgard, P. (1998). Defining modules, modularity and modularization. *Proceedings of the 13th IPS research seminar*, Aalborg, Denmark.

Mocker, M. and Boochever, J. O. (2020). How to Avoid Enterprise Systems Landscape Complexity. *MIS Quarterly Executive*, 19(1).

Orlikowski, W. J. and Gash, D. (1994). Technological Frames: Making Sense of Information Technology in Organizations. *ACM Transactions on Information Systems*, 12(2), 174-207.

Papazoglou, M. P. and Van Den Heuvel, W. J. (2007). Service oriented architectures: Approaches, technologies and research issues. *The VLDB Journal*, 16(3), 389-415.

Ross, J. W., Beath, C. M. and Mocker, M. (2019). "Creating digital offerings customers will buy." *MIT Sloan Management Review*, 61(1), 64-69.

Saltzer, J. H., Reed, D. P. and Clark, D. D. (1984). End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4), 277-288.

Saunders, M., Lewis, P. and Thornhill, A. (2015). *Research methods for business students*. London: Pearson Education.

Schilling, M. A. (2000). Toward a General Modular Systems Theory and Its Application To Interfirm Product Modularity. *Academy of Management Review*, 25(2), 312-334.

Schmid, M., Haki, K., Tanriverdi, H., Aier, S. and Winter, R. (2021). Platform over Market – When is joining a Platform Beneficial? *29th European Conference on Information Systems*, Marrakech, Morocco.

Schmidt, C. (2015). *Business Architecture Quantified: How to Measure Business Complexity. In Business Architecture Management.* Springer, Cham.

Schmidt, C. and Buxmann, P. (2011). "Outcomes and success factors of enterprise IT architecture management: empirical insight from the international financial services industry." *European Journal of Information Systems*, 20(2), 168-185.

Schneider, A. W., Zec, M. and Matthes, F. (2014). Adopting Notions of Complexity for Enterprise Architecture Management. *20th Americas Conference on Information Systems*, Savannah, USA.

Schütz, A., Widjaja, T. and Kaiser, J. (2013). Complexity in enterprise architectures: conceptualization and introduction of a measure from a system theoretic perspective. *21st European Conference on Information Systems*, Utrecht, Netherlands.

Shapiro, C. and Varian, H. R. (1999). The art of standards wars. *California Management Review*, 41(2), 8-32.

Tassey, G. (2000). "Standardization in technology-based markets." *Research Policy*, 29(4-5), 587-602.

Teixeira, M., Ribeiro, R., Oliveira, C. and Massa, R. (2015). A quality-driven approach for resources planning in Service-Oriented Architectures. *Expert Systems with Applications*, 42(12), 5366-5379.

Tiwana, A. and Konsynski, B. (2010). "Complementarities between organizational IT architecture and governance structure." *Information Systems Research*, 21(2), 288-304.

Van Oosterhout, M., Waarts, E. and Van Hillegersberg, J. (2006). Change factors requiring agility and implications for IT. *European Journal of Information Systems*, 15(2), 132-145.

Widjaja, T. and Gregory, R. W. (2020). Monitoring the Complexity of IT Architectures: Design Principles and an IT Artifact. *Journal of the Association for Information Systems*, 21(3), 4.

Widjaja, T., Tepel, D., Kaiser, J. and Buxmann, P. (2012). Heterogeneity in IT Landscapes and Monopoly Power of Firms: A Model to Quantify Heterogeneity. *33rd International Conference on Information Systems*, Orlando, USA.

Xia, W. and Lee, G. (2005). "Complexity of Information Systems Development Projects: Conceptualization and Measurement Development." *Journal of Management Information Systems*, 22(1), 45-83.