DESIGNING FOR INNOVATION BY APPLYING ORGANIZATIONAL MODULARITY

Philip Huysmans
Department of Management Information Systems, University of Antwerp, Prinsstraat 13, Antwerp, Belgium
philip.huysmans@ua.ac.be

Keywords: Enterprise Architectures, Business Innovation and Software Evolution, Business-IT Alignment and Traceability.

Abstract: In volatile and customer-driven markets, the ability to innovate is a key success factor. Innovations have to be implemented at a steady pace to ensure business sustainability. However, the innovation process is only poorly understood. Therefore, many organizations and governments have difficulties stimulating and managing innovation. Several authors have proposed organizational modularity as a theoretical basis to understand and manage innovation. Their main argument is that a modular structure enables parallel evolution of different organizational modules. Consequently, innovations can be implemented without being limited by implementation aspects of other organizational modules. Modularity has been applied by various authors on different levels of the organization, such as products, processes, departments, and supporting IT systems. Moreover, enterprise architecture frameworks allow the modeling of different viewpoints of an organization. However, few organizations can be considered to have completely decoupled organizational layers. Consequently, dependencies between organizational modules on different enterprise architecture layers can heavily impact the ability to introduce innovations. In this paper, we demonstrate how modular dependencies impact enterprise architecture projects in two case studies. We then present a case study to illustrate how a modular dependency approach can be used to complement existing modeling approaches.

1 INTRODUCTION

Contemporary organizations are faced with rapidly changing environments. As a result, innovations need to be introduced in the organization in order to remain competitive in these markets. The introduction of innovations often impacts many different aspects of the organization. For example, Barjis and Wamba describe the impact on organizations caused by the introduction of RFID technology (Barjis and Wamba, 2010). Besides the adaptation of the business processes through, for example, BPR, it can be expected that other organizational artifacts need to be adapted as well. Data management systems need to able to handle "enormous" data volumes (Barjis and Wamba, 2010), hardware infrastructure and software applications need to be purchased to handle accurate tracking, and customer acceptance needs to be handled by privacy commissions and marketing departments.

Enterprise architecture projects are frequently initiated to guide the change process needed to implement such innovations (Schekkerman, 2005). Enterprise architecture frameworks help to identify the different elements to which changes need to be applied. In such frameworks, models are created from different perspectives or viewpoints. Complexity from other viewpoints is abstracted away to be able to focus on the concerns of a specific viewpoint. While this approach aids understandability of different aspects of the organization, it can lead to unexpected results when artifacts from different viewpoints impact each other. This can be observed in the case of RFID, where changes to processes can be hindered by the data structures on which they operate. When innovative processes are designed using a BPR approach, but data systems are unable to support the required data types, implementation of the new processes will be problematic. Possibly, the redesigned processes will then need to be altered in order to be supportable. Consequently, adapting processes can be impacted by the concrete data implementation, which is not completely visible in the models which are created in a

Huysmans P.
DESIGNING FOR INNOVATION BY APPLYING ORGANIZATIONAL MODULARITY.
DOI: 10.5220/0004458500510060
In *Proceedings of the First International Symposium on Business Modeling and Software Design (BMSD 2011)*, pages 51-60
ISBN: 978-989-8425-68-3

process viewpoint. While aspects of this implementation are not visible in these models, they do impact the way these models can be used, and therefore need to be considered.

In this paper, we argue that *modular dependencies* can be used to explicitly identify such impacts. This approach can be applied complementary to existing enterprise architecture modeling. We introduce relevant literature for this approach concerning modularity and enterprise architecture frameworks in Section 2. We then present three case studies to illustrate the application of modular dependencies in organizations which are faced with innovations which require changes in several viewpoints. In Section 3, we present two case studies to demonstrate the importance of modular dependencies between layers in the enterprise architecture. In the first case study, we describe how dependencies between enterprise architecture layers limit the ability to react to market changes in a public broadcasting company. In the second case, we elaborate on a previously published case study to show the importance of eliminating such dependencies in order to design a structure which allows the implementation of innovations. In Section 4, we demonstrate in more detail how an approach using modular dependencies can be applied during an enterprise architecture project. Finally, we present our conclusions on these cases in Section 5.

2 RESEARCH BACKGROUND

In this section, we introduce a necessary background of modularity and enterprise architecture research literature.

2.1 Modularity

Organizational modularity recently receives much attention in both research and practice. Campagnolo and Camuffo provide a literature overview of 125 management studies which use the modularity concept (Campagnolo and Camuffo, 2010). They define modularity as "an attribute of a complex system that advocates designing structures based on minimizing interdependence between modules and maximizing interdependence within them" (Campagnolo and Camuffo, 2010). They argue that organizational artifacts such as products, production systems, and organizational structures can be regarded as modular structures. A characteristic of a perfect modular structure is that it allows parallel evolution of different modules (Baldwin and Clark, 2000). However, dependencies between the modules of such a structure limit the autonomy of the individual modules. Baldwin and Clark state that "[b]ecause of these dependencies, there will be consequences and ramifications of any choice" made during the design of the artifact (Baldwin and Clark, 2000). A design choice for a given parameter can limit or affect the possible design choices concerning other parameters. In traditional modularity approaches (e.g., product modularity), dependencies between and within modules are visualized by Design Structure Matrices (DSM). In a DSM, an artifact is described by a set of design parameters. The matrix is then filled by checking for each parameter by which other parameters it is affected and which parameters are affected by it. The result is a map of dependencies that represent the detailed structure of the artifact. An example design structure matrix is shown in Figure 1. Dependencies are represented by an "x". The intersection of identical design options is marked with a ".". Consider the design dependency which is represented by the "x" in the intersection of the column of design option A2 and the row of design option A1. This signifies that design option A2 influences design option A1: the design decision for design option A1 will be dependent on the decision taken for design option A2. This dependency does not break the modular structure of the artifact, since design options A1 and A2 both belong to the same module. Now consider the dependency of design option B1 on design option A2. Since these design options belong to different modules, it can be concluded that these modules are directly dependent on each other. Therefore, this dependency does violate the modular structure. Indirect or chained dependencies can occur as well. While design option B2 does not seem to affect any design options of module A, it does affect design option B1. As we discussed before, design option B1 does affect design option A1. Therefore, a so-called chained dependency exists between design options B2 and A2.

		Module A		Module B	
		Option A1	Option A2	Option B1	Option B2
Module A	Option A1	.	x		
	Option A2		.	x	
Module B	Option B1			.	x
	Option B2				.

Figure 1: An example Design Structure Matrix.

Originally, DSMs were used to model modular products. In later publications, a DSM was also applied to model organizational departments (Baldwin and Clark, 2003). This indicates that similar tools can be used for analyzing modularity on various levels. In this paper, we will focus on modularity on the organizational and software levels. Organizational modularity focuses on other artifacts than product modularity within the same organization. Research on this level has been performed by, for example, Galunic and Eisenhardt (Galunic and Eisenhardt, 2001). Galunic and Eisenhardt consider organizational divisions as modular organizational building blocks. These divisions, which have independent decision power, cost structures and profit responsibility, are a combination of capabilities and charters. Charters represent the task, market and customer a division is concerned with. Charters need to be able to change as markets evolve. By dynamically attributing these charters to organizational divisions, a flexible organization is created which can adapt to changing market conditions. This kind of modularity therefore enables the flexibility required on a business level. On the software level, modularity is used to achieve a flexible structure as well. For example, Parnas argued that a modular decomposition in software systems should be made to isolate the impact of changes (Parnas, 1972). When the impact of a change remains within a module, changes can be applied to individual modules without requiring changes in the rest of the system. While modularity is applied to both the organizational and IT level by various researchers, most research project focus on a single level. However, interactions between a modular approach on the organizational and the software level remain an important issue. Given the high dependence on IT systems, it is important that a change on the organizational level (e.g., changing chapters) can be handled by the supporting IT systems as well. Otherwise, the inability to change the IT systems will restrict the ability to change organizational modules. Indeed, modularity needs to be considered as a *relative* attribute of complex systems (such as organizations), meaning that within a single artifact, different levels of modularity can exist (Simon, 1962).

2.2 Enterprise Architecture

Enterprise Architecture (EA) frameworks provide insight to the structure of organizational goals, divisions, and supporting IT systems. By specifying separated viewpoints on organizational artifacts, an overview is provided of specialized models created by different stakeholders (The Open Group, 2009). Most enterprise architecture frameworks use a top-down perspective. They start by defining business goals and a high-level artifacts to realize these goals (e.g., an organizational structure). Based on these artifacts, lower-level artifacts are defined which offer services to support the business level. The following architectural levels are usually identified (Schöenherr, 2008):

- Business layer
- Functional layer
- Information layer
- Infrastructure layer

For example, TOGAF suggests to use an iterative process consisting of eight phases to develop an enterprise architecture. Based on the business goals which are defined in the first phase, different architectures need to be developed in the following order: business architecture (second phase), information systems architecture (third phase), and technology architecture (fourth phase). These architecture correspond to the defined layers. The business architecture is defined on the business layer. The information systems architecture consist of both the functional and information layer. The technology architecture is addressed on the infrastructure layer. This approach assumes that supporting services can be created straightforwardly based on business requirements. However, when implementation-focused approaches are described (e.g., Service-Oriented Architecture), the existing infrastructure is often an important restriction on the services which can be provided. Consequently, the focus of such approaches is often on subjects such as legacy integration. Approaches which explicitly integrate an implementation approach with business goals (e.g., SOMA) therefore use a middle-out approach (Arsanjani et al., 2008). Such an approach takes into account the possibilities of the supporting systems, and attempts to find the best solution for the business requirements. Indeed, the use of additional methods illustrates that a top-down approach in enterprise architectures needs complementary approaches to deal with complex environments. In a literature review on enterprise architectures, Lucke et al. identify several issues which motivate this need (Lucke et al., 2010). First, *complexity* is referred to as an underestimated issue. Not only the complexity of the models themselves, but also the dependencies between the different layers remain problematic. Second, *rapidly changing conditions* imply that a top-down specification of an enterprise-wide architecture can become out-dated before it is even implemented. Third, a top-down specification of the architectural layers results in issues regarding *scoping of architecural descriptions*. Rather than being straightforward, the identification of organizational and technical services to support business requirements is often considered to be problematic. Therefore, a complementary approach is required to represent the impact of lower-level layers on higher-level layers.

2.3 Modularity in Enterprise Architectures

Several authors link the explicit decomposition of viewpoints in enterprise architecture frameworks to the ability to independently change artifacts. This indicates how enterprise architecture frameworks can be linked to the modularization of organizations. For example, business modularity is considered to be the highest level of enterprise architecture maturity (Ross and Beath, 2006). On this maturity level, the role of IT in an enterprise architecture is to "provide seamless linkages between business process modules" (Ross and Beath, 2006). Such business process modules allow "strategic experiments that respond to changing market conditions" (Ross and Beath, 2006). Based on a practitioners survey, it seems that many business users indeed expect an enterprise architecture to enable their ability to change in response to market conditions (Schekkerman, 2005). A modularity perspective can aid to specifically focus on the issues specified by Lucke et al. (Lucke et al., 2010), which have been mentioned above: identifying modular dependencies reveals how complexity is introduced when modules are added; the parallel evolution of modules limits the impact of rapidly changing conditions; the explicit specification of modules aids scoping architectural descriptions.

Consequently, independence between enterprise architecture layers should be achieved by defining business components and standardized interfaces based on the artifacts modeled in the different layers. This shows that an explicit focus on the coupling of artifacts from different enterprise architecture layers is required to gain insight in the kind of changes which can be supported. In the case studies we performed, dealing with this kind of coupling adequately often seemed to be an important success factor. An approach which explicitly shows these impacts can therefore help to improve insight in the change process.

3 CASE STUDY OBSERVATIONS

In this section, we illustrate the occurrence and impact of modular dependencies between artifacts from different levels in an enterprise architecture. The presented case studies are part of a larger group of case studies which aim to apply insights from modularity to enterprise architectures. These case studies have been performed adhering to the case study methodology (Yin, 2003). Given the ambiguity of the definition of enterprise architecture in both practice and academic literature, and the large scope of enterprise architecture frameworks, it is difficult to clearly distinguish between the research subject itself and the organizational environment. Therefore, we selected an exploratory case study approach, since it is well suited for research goals where the boundaries between phenomenon and context are not clearly defined (Yin, 2003). In the various cases, we have used the key informant method to identify informants who were highly knowledgeable about and involved in the enterprise architecture projects. The primary mode of data collection consisted of face-to-face interviews. In preparation for these interviews, various documents (e.g., documentation and presentation materials, documents on the organizational structure) have been consulted to gain an initial understanding of the organization and the project itself. A case study protocol was crafted, including an initial set of questions. These questions concerned various topics (e.g., architecture definition, expected benefits and barriers), in order to obtain a thorough description of the project. Follow-up questions took place via e-mail. During the interview, additional sources of evidence were collected, such as articles and internal documentation. The interview was digitally recorded and transcribed for future reference.

3.1 Public Broadcasting Company

In a first case study, we observe a public broadcasting company (PBC) which is faced with changing customer demands. We focus on the division which is responsible for broadcasting news journals. Traditionally, this organization offers radio and television transmissions, which follow a clearly defined schedule. The radio and television business units import news items from different sources, such as feeds from external agencies, or items made by reporters. Items can be imported using physical tapes, digital files or through satellite transfers. The items are then edited and transmitted in the form of a news journal. However, since the introduction of the internet, customers demand personalized and real-time access to transmissions. Therefore, the content of news journals needs to be approached differently. The PBC decided to create a dedicated business unit to create an online channel, next to the existing radio and television units. This new business unit could reuse content from both the radio and television units, and create dedicated online news items as well. Adding the online channel was considered to be a necessary strategic move in order to serve a changing market. For the PBC, the radio, television and online business units are therefore situated on the *business layer* of their enterprise architecture.

Adding this additional business unit posed serious problems due to the supporting structure of the PBC on the lower enterprise architecture levels. For example, accessing existing radio and television items proved to be complex, since they were stored in specialized applications. For every import channel (tape, digital or satellite), different applications needed to be used. This is a direct result of the principles used to develop the application portfolio, which was located on the *functional layer* in the enterprise architecture. As a general principle, the organization always selects best-in-class software solutions for specialized media editing. This principle ensured the most efficient editing process. However, this results in an application portfolio which is not well integrated. Employees with specialized competences are required to operate these software packages. Therefore, in order to reuse audio and video fragments for the online channel, employees with these competences needed to be made available for the online business unit. This resulted in a duplication of skill sets, which was not efficient. The inclusion of employees with the competence to use the specialized software packages was not foreseen when the new business unit was defined. However, the coupling between software packages and employee competence on the information layer necessitates this inclusion. Put differently, the required competences defined on the business layer need to be adapted to account for a dependency on the functional level. It should be noted that coupling on the functional layer is solved by adapting the artifacts which are defined on the business layer. Consequently, the ability to take strategic decisions to serve an emerging market are impacted by decisions made on the functional layer. While the principle to select best-in-class applications may be justified for this sector and the performance of the organization, the lack of integration which it causes and the restrictions it places on business flexibility need to be understood as well. When considered as a modular structure, this dependency can be represented using a DSM. The DSM is presented in Figure 2. It shows how a design parameter from the applications, which are part of the functional layer, affect the organization chart design parameter of the business unit, which is part of the business layer.

Since the PBC was unable to motivate the costs of specialized employees for the new channel, a structural solution was proposed. The dependency on the

		Business unit		Application	
		Organization chart	Charter	Competence	Platform
Business unit	Organization chart	.		x	
	Charter		.		
Application	Competence			.	
	Platform				.

Figure 2: The DSM for PBC.

required application capabilities forces the the organization to deal with concerns from the functional layer on the business layer, i.e., the different handling of tape, digital and satellite items. This dependency was removed by developing an abstraction layer on top of the functional layer, which provided only the functionality needed on the business layer. This abstraction layer was defined based on concepts known in the business layer: the basic entity on this abstraction layer was a news item. A news items entity abstracts from the method which was used to import the source material. Therefore, it served as a kind of interface to the specialized software applications. For every application, audio and video fragments can be extracted, together with the required meta-data, in a uniform way. The specific functionality of the editing programs is not available through this interface. In order to use the editing functionalities, specialized competences remain necessary. However, the need for employees with these competences is now not imposed on the business layer, only on the functional layer.

This introductory case shows that decisions taken on the functional layer of enterprise architectures can make changes to the business layer more complex. By adequately encapsulating the complexity from the functional layer, dependencies between the different layers can be removed, and necessary services can still be offered. As a result, decisions on the business layer only need to deal with the inherent complexity imposed by the business environment, instead of dealing with complexity originating from the supporting layers as well. In this case, the number of modules is low and the dependency which limits the business layer is clear. However, this case provides a clear illustration of the issue we address in this paper.

3.2 Gas Flow Manager Company

In a second case study, we observed an enterprise architecture project at a gas flow manager company. The company offers gas transport services to its customers based on a grid consisting of entry points, nodes, and pipelines connecting the nodes. The functional and information layer of the enterprise architecture had to be rebuilt after the company was separated from a gas trading company. The legislation concerning liberalization of the energy market demanded this separation, as the company had to offer its gas transport services to other gas trading companies as well. Prior to the architectural redesign, IT was generally considered to be a bottleneck during the implementation of changes by business users. The new architecture needed to be able to respond better and more quickly to changing business requirements and had to be understandable for business users, so they could more realistically estimate the impact of the changes they requested. Therefore, it was decided to clearly align the functional architecture with a high-level enterprise architecture model on the business level. This model needed to be constructed in such a way that the *stable* operation and stakeholders of the organization are represented. If the subsequent changes required changes in the high-level model, it would not be useful as a basis for the functional architecture. The issue of rapidly changing conditions has indeed been addressed as an important enterprise architecture issue by Lucke et al. (Lucke et al., 2010). Put differently, business changes need to be attributed to the implementation of the model elements, not on the nature of the elements itself. The model which was designed is shown in Figure 3. The modeled activities represent generic descriptions of the business-level construction of the organization (e.g., *Define commercial services*). In order to achieve a well-aligned business and functional architecture, a separate application has been developed to support the scope of exactly one activity in this model.

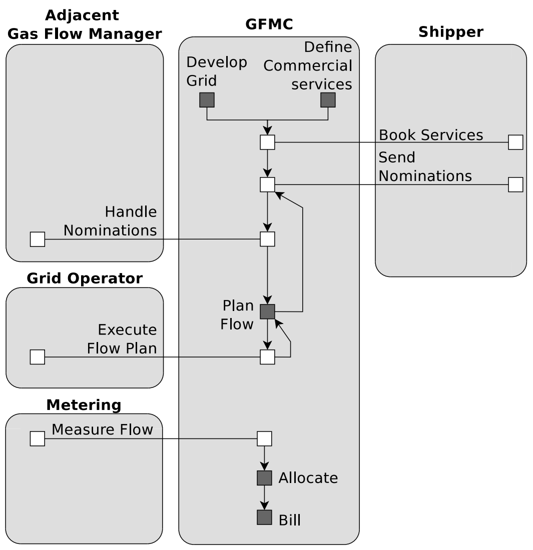

Figure 3: Stable GFMC model.

In a previous publication, we focused on a repeatable and reproducible method to design such models (Huysmans et al., 2010), and the benefits which can be achieved with such an approach. That description was limited to the top layers of the enterprise architecture, i.e., the business and functional layers. In this paper, we focus on the preconditions for applying this approach, which are situated on the functional and information layers. For this approach to work, the specification the functional layer may not be restricted by the information layer. Modular dependencies between artifacts from the information layer and artifacts from the functional layer can be used as an indication for such restrictions. Before the organizational split, applications had separate data sources, which impacted the specification of application scope. Put differently, dependencies existed between the functional and information layers. Because the semantic differences between entities in these data sources, it was not feasible to design applications which relied on data from different sources without creating complexity within the application. In the enterprise architecture maturity model, this indicates an architecture which is not correctly modularized (Ross and Beath, 2006). For example, the entity *"customer"* existed in different data sources. In some sources, this entity referred to end consumers of the gas which is transported. In other sources, it contained the customers of the gas flow manager company (i.e., the gas traders). As a result, the scope of applications was defined based on the scope in the data sources. During the enterprise architecture project, a glossary was developed iteratively with business users from different areas to ensure consistent terminology across the organization. The glossary defines the entities and their relationships to each other. Based on this glossary, a database scheme was developed and imposed on the different data sources. Consequently, the dependency of the application scope on the data source scope was resolved. The coupling between the functional and information layer was thus removed. As a result, the modules on the functional layer could be designed to be well-aligned with the business layer, without being restricted by aspects from the information layer.

Moreover, the solution applied in this case follows the solution presented in modularity theory. In order to resolve modular dependencies, modularity theory proposes to specify an *architectural rule*. Such a rule limits the design freedom for the implementation aspects of modules by prescribing a certain design choice. In this case, the glossary limits the allowed interpretation of, for example, the *"customer"* entity. As a result, no conversions need to be performed to ensure a correct use of data entities. Other solutions are possible to resolve dependencies as well. Currently, we focus on the ability of modular dependencies to identify coupling between modules of different architectural layers as a cause for restrictions on higher-level layers. In future research, we elaborate on different methods to deal with this kind of coupling. However, the current case illustrates how a modularity approach can be complementary to enterprise architecture frameworks. While the different applications and data sources can be represented in an enterprise architecture, no indications of the coupling between the different viewpoints can be represented. When we consider the applications and data sources as modules, we can use a Design Structure Matrix (DSM) to represent aspects of the module implementation which affect each other.

4 APPLICATION OF MODULAR DEPENDENCIES

In the previous cases, we illustrated the relevance of modular dependencies in enterprise architecture projects. We now apply the insights gained from the analysis described above to a case study in a governmental organization. The mission of the organization is to introduce and implement e-government solutions. To achieve this goal, it undertakes projects in the field of back-office reengineering, and tries to leverage these improvements by supporting projects with governmental partners. In this paper, we focus on a project that improves the way data from various sources is used in governmental processes. We will refer to this project as the Data Usage in Governmental Processes (DUGP) project. Similar to the description in Section 3.2, the structure of the data sources limits the design of governmental processes. Because of the political situation, different data sources are controlled by different governmental entities, which belong to governments on different levels (e.g., federal, regional, local level). As a result, different implementations exist for a large amount of design parameters. For example, the data delivery design parameter may be implemented by an online web interface, an FTP transfer, or through web services. Consider the partial Design Structure Matrix represented in Figure 4, which has been developed to describe the situation before the DUGP project. In this DSM, multiple design parameters are considered simultaneously. Compare this to the coupling in the GFMC enterprise architecture, where we focused on a single design parameter (i.e., data semantics).

The “x”-es with the grey background represent (1) the dependency of the data retrieval design parameter of the processes on the data delivery design parameter of the data sources, and (2) the dependency of the data syntax used in the processes on the data syntax used in the data sources. Consider the impact of these two design parameters in the following example. A process to request construction premiums requires personal data of the citizen requesting the premium (from data source A) as well as geographical data of the construction site (from data source B). Since the databases which contain the personal and geographical data are not integrated or standardized, various conversions between the implemented data parameters of these data sources may be necessary. Suppose that the information required from data source A needs to be obtained by invoking a single web service call, providing the address from the end user using four data fields (street name, street number, bus number, city name). In order to query geographical information in data source B, a request file containing two data fields (street name and street number, and ZIP code) has to be transferred using the FTP protocol. Since the construction premium process depends on both data sources, it needs to be able to communicate using two different versions of address data syntax, and two different versions of data retrieval technology. If the construction premium process needs to be changed, and an additional data source is required, the process owner needs to be aware of the data syntax and data retrieval method offered by the new data source. Moreover, when changes are made to these implementation aspects of data sources, additional versions need to be supported by the processes. The resulting complexity of these conversions is a barrier for the use of these data sources. Moreover, this example shows that different design parameters are intertwined in a certain implementation. As a result, it is hard to resolve these dependencies individually. The goal of the DUGP project is therefore

		Process				DS			
		Throughput	Data Retreival	Data Syntax	Data Dictionary	Capacity	Data Delivery	Data Syntax	Data Dictionary
Process	Process Throughput	.	x			x			
	Data Retrieval		.	x		x	x		
	Data Syntax			.	x			x	x
	Data Dictionary			x	.			x	x
DS	Capacity					.	x		
	Data Delivery					x	.	x	
	Data Syntax						x	.	x
	Data Dictionary							x	.

Figure 4: DSM before the DUGP project.

to eliminate these dependencies at once in order to reduce the complexity of using data sources in governmental processes. However, the solution which is suggested by modularity theory, i.e., the definition of architectural rules, was not feasible because of the governmental structure. The different data sources are controlled by different governmental entities, who can decide independently on the implementation of design parameters. Declaring an architectural rule for a design parameter therefore requires an agreement between all governmental units responsible for a data source. However, since most units rely heavily on legacy systems to provide data services, changes to the implementation of design parameters are not easily realized. Therefore, it is difficult to reach such an agreement if an organization which can impose rules to these governmental units is not in place.

Consequently, the e-government organization developed a platform to consolidate data sources and provide uniform data access. Similar to the PBC case discussed in Section 3.1, an abstraction layer was developed to offer the required functionality without exposing the complexity of the layer offering these services. This abstraction layer provides services from the information layer to governmental processes, which are considered to be on the functional layer. The platform is based on two existing data sources from the federal government. Data from these data sources will be augmented with data available in data sources from other governments (e.g., geographical data, which is offered by regional governments). The first data source which is used focuses on data concerning organizations. In this data source, data such as registration number, official addresses and legal statute of enterprises can be obtained. We will refer to this data source as the Data Source for Organizations (DSO). The Federal Public Service Economy is responsible for this data source. The second data source offers data concerning individuals. It refers to data such as employment and social status of citizens. We will refer to this data source as the Data Source for Individuals (DSI). This data source is governed by a separate organization created by the federal government. The platform will maintain this distinction, and offer its data services grouped in an Enhanced Data Source for Organizations (EDSO) and an Enhanced Data Source for Individuals (EDSI).

Since the EDSO relies on the DSO for its data, and the EDSI relies on the DSI, they need to consider the implementation of these data sources. Many implementations of design parameters are quite different. For example, the DSI has webservices available to query its data. As a result, these webservices can be used to develop webservices in the EDSI. These webservices are not directly offered in their original form. Instead, a facade pattern is used. This enables the creation of a uniform web service syntax throughout the platform. Otherwise, a dependency on the data syntax design parameter would be introduced. In contrast, the DSO has no webservices available. It is a mainframe which operates using batch requests. Therefore, a copy is made from the original DSO every night. This copy is then augmented with data from other governmental authorities, and used as a central database on which the services from the EDSO are provided. In order to simplify data access, the new platform provides three data delivery methods which will be available for all data sources: data repositories, an online application and webservices. Customized data repositories are large data files, which are copied to the process owner. After this initial data provision, automatic updates are transferred when data changes. These repositories are offered to enable process owners to incorporate the data from the new platform in their processes, without having to implement a webservices-based data access. Since many organizations are accustomed to using their own data sources in their processes, a customized data repository can be implemented without many changes in the processes. However, the unauthorized data sources which have been collected by the organizations themselves will then be replaced with authentic data. The online application allows for manual consultation of the data with a much smaller granularity: instead of a single large data file, only the result of a single query is returned. The same result can be obtained automatically through the use of webservices. Webservices offer the same data granularity, but can be implemented to automate processes.

From a modularity perspective, the platform can be considered as an additional module to eliminate dependencies between data and process modules. The DSM for the DUGP project is shown in Figure 5. When comparing DSM of the platform in Figure 5 with the DSM in Figure 4, we can conclude that some of these dependencies are indeed eliminated. Consider for example the data syntax and data delivery. We included an empty grey background to mark the previous existence of these dependencies. The syntax of webservices offered by EDSI is decoupled from the naming conventions of the DSI by using a facade pattern. As a result, naming conventions can be kept internally consistent with custom-built webservices for EDSO. The data syntax can be considered as an architectural rule which is maintained by the e-government organization. By adhering to this data syntax, process owners no longer need conversions between different data syntax versions. Another example is the data de-

		Process				Platform				DS			
		P. Throughput	Data Retrieval	Data Syntax	Data Dictionary	Capacity	Data Delivery	Data Syntax	Data Dictionary	Capacity	Data Delivery	Data Syntax	Data Dictionary
Process	Process Throughput	.	x			**x**							
	Data Retrieval	x	.	x		x					▨		
	Data Syntax	x		.	x				x			▨	
	Data Dictionary				.				x				x
Platform	Capacity					.	x			x	x		
	Data Delivery					x	.	x		x			
	Data Syntax						x	.	x			x	
	Data Dictionary				x			x	.			x	x
DS	Capacity									.	x	x	
	Data Delivery									x	.	x	
	Data Syntax										x	.	x
	Data Dictionary											x	.

Figure 5: DSM of the DUGP project.

livery design. In data sources from the platform, data can be provided through customized data repositories, an online application or webservices. Here, a single design option has not been selected, but the consistent offering of all data delivery techniques allows process owners to implement a single design for data delivery. Again, process owners no longer depend on the specific data delivery technique of the individual data sources. Consequently, it seems that the platform aids to decouple the process owners from the design decision of the data sources.

However, as stated by Baldwin and Clark, eliminating all dependencies in a modular structure is not a trivial task (Baldwin and Clark, 2000). Consider the design option *process throughput* in the case of an automated process. Our respondents indicated that the number of processes which can be supported is limited by, amongst others, the capacity of the data delivery implementation. In Figure 5, this is visualized by the "x" where the column of the capacity of the platform intersects with the row of process throughput. A possible data delivery implementation in the platform are webservices. As described above, webservices from the platform can be either custom-built by the e-government organization (e.g., webservices for EDSO), or can be part of a facade-pattern, calling underlying webservices (e.g., webservices for EDSI). The custom-built webservices operate on a local database. Consequently, their capacity is limited by the servers of the e-government organization itself. However, webservices which are part of the facade pattern are dependent on the capacity of the underlying services of DSI. Based on the implementation, issues with webservice capacity need to be discussed with the e-government organization or with the organization responsible for the original data source. The difference between the implementation of webservices in the platform is based on the available data delivery techniques from the original data sources. In Figure 5, this is visualized by the "x" where the column of the data delivery of a data source intersects with row of the capacity of the platform. It therefore seems that an unexpected dependency can be identified: when the platform is used, the process throughput design decision is impacted by the data delivery technique of the original date source. When the data delivery of DSO is changed (e.g., webservices become available) and used by the platform, process performance may be impacted. This is an example of a chained design dependency which propagates through the design structure matrix. Such dependencies are difficult to trace and to account for in change projects.

5 CONCLUSIONS

In this paper, we explored a concrete application of modularity on the organizational level. We have shown that by modeling modular dependencies, interactions between layers in enterprise architecture models can be represented. Such interactions are not explicitly focused on in enterprise architecture frameworks. Therefore, this approach is complementary to existing frameworks. These frameworks usually focus on the top-down specification of different viewpoints. However, these viewpoints can not be considered to be independent from each other in complex organizations. As a result, one needs to be aware of the impacts and restrictions imposed by lower-level layers during a top-down specification of enterprise architecture models. Therefore, research on this subject should be based on observations in real-life case studies instead of on theoretical examples. We focused on the restrictions of modeling artifacts on higher-level layers based on dependencies on the implementation of design parameters of lower-level artifacts. First, we demonstrated how this effect occurs in the PBC case study. Second, we discussed how the elimination of such dependencies can be a prerequisite in successful enterprise architecture projects. We illustrated this prerequisite in the context of the GFMC case study, which was published earlier. Moreover, we explored the applicability of dealing with modular dependencies as suggested by modularity literature. This solution implies the definition of architectural rules to limit the implementation possibilities of design parameters. Consequently, artifacts which are dependent on these design parameters can assume that a fixed implementation will always be supported. Third, we applied the insights from the observations in these case studies more concretely to the DUGP project. We showed that a DSM can be used to represent relevant issues for the enterprise architecture project as modular dependencies. This perspective allows to objectivate the issues which are resolved by the project. Moreover, a structured analysis can lead to the discovery of remaining issues after the project. Remaining issues can be unresolved dependencies, or newly introduced dependencies. This was illustrated by identifying the *capacity* design parameter as a chained dependency.

Future research needs to be conducted to gain insight on how modular dependencies on this level can be dealt with. In some cases, the definition of architectural rules seems appropriate. However, instead of choosing a single implementation option, it has already been observed that a consistent offering of multiple implementations can be required as well. Moreover, in some cases, imposing architectural rules does not seem feasible, because of the organizational structure. In the DUGP project, it was impossible to impose architectural rules to different organizational units. Therefore, an additional module was added. However, the introduction of new dependencies shows that a structured approach is required to adequately resolve modular dependencies.

REFERENCES

Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Gariapathy, S., and Holley, K. (2008). Soma: a method for developing service-oriented solutions. *IBM Syst. J.*, 47:377–396.

Baldwin, C. Y. and Clark, K. B. (2000). *Design Rules, Volume 1: The Power of Modularity*, volume 1 of *MIT Press Books*. The MIT Press.

Baldwin, C. Y. and Clark, K. B. (2003). The value, costs and organizational consequences of modularity. Working Paper.

Barjis, J. and Wamba, S. F. (2010). Organizational and business impacts of rfid technology. *Business Process Management Journal*, 16(6):897–903.

Campagnolo, D. and Camuffo, A. (2010). The concept of modularity in management studies: A literature review. *International Journal of Management Reviews*, 12(3):259–283.

Galunic, D. C. and Eisenhardt, K. M. (2001). Architectural innovation and modular corporate forms. *The Academy of Management Journal*, 44(6):1229–1249.

Huysmans, P., Ven, K., and Verelst, J. (2010). Designing for innovation: using enterprise ontology theory to improve business-it alignment. *Proceedings of the 1st International Conference on IT-enabled Innovation in Enterprise*, pages 177–186.

Lucke, C., Krell, S., and Lechner, U. (2010). Critical issues in enterprise architecting - a literature review. In *AMCIS 2010 Proceedings*.

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058.

Ross, J. and Beath, C. M. (2006). Sustainable it outsourcing success: Let enterprise architecture be your guide. *MIS Quarterly Executive*, 5(4):181–192.

Schekkerman, J. (2005). Trends in enterprise architecture: How are organizations progressing? Technical report, Institute For Enterprise Architecture Developments.

Schöenherr, M. (2008). Towards a common terminology in the discipline of enterprise architecture. In *ICSOC Workshops'08*, pages 400–413.

Simon, H. A. (1962). The architecture of complexity. *Proceedings of the American Philosophical Society*, 106(6):467–482.

The Open Group (2009). The open group architecture framework (togaf) version 9. http://www.opengroup.org/togaf/.

Yin, R. K. (2003). *Case Study Research: Design and Methods*. Sage Publications, Newbury Park, California, 3rd edition.