Applying Axiomatic Design and Conceptual Independence in the Domain of IT Systems

Debbie Tarenskeen* and *René* Bakker

HAN University of Applied Sciences, Arnhem and Nijmegen, The Netherlands

Abstract. In this paper we explain why Axiomatic design has not been applied in large system of systems information technology architectures in health care organizations in the Netherlands. We have found in extensive case studies of IT systems that the Independence axiom could not be found in the existing Information systems. This causes great concern for the adaptability of systems. Although best practices in system engineering advice decoupling of system functionality, findings show that the Independence Axiom has not been applied to functional requirements. A number of difficulties was exposed to the researcher and IT architects, when they tried to identify and to demarcate functional requirements in existing systems. In IT systems a distinction is made between business and system functions and software engineers emphasize decoupling of system functions. That has resulted in a strong coupling of business information and technical application logic. This means Conceptual dependencies are often at the heart of the difficulties. Decoupling these dependencies in applications seems a candidate for ordering requirements in a way that keeps the functional requirements independent from each other. Conceptual independence signifies a complete decoupling of the concepts and relations of the Business domain from the technical functionality. The paper describes the difficulties when applying the Independence Axiom in the Information systems domain and argues that a combination of Conceptual independence and Axiomatic design is achievable. The authors adapt the Design matrix with Conceptual independence. We conclude that applying Conceptual independence is crucial when constructing IT systems based on Axiomatic design.

1 Introduction

In this paper we will argue that Axiomatic design cannot be applied in IT systems consisting of many subsystems, because of Conceptual dependencies. In a paper of Suh in 1998, he argues that Axiomatic design can be applied in system design for different kinds of systems, including systems for Information technology [1]. The authors found in three case studies that IT systems were hard to change. They examined the IT systems and detected that Axiomatic design had not been applied. In the next chapter we will outline the three case studies in health care. In chapter 3 we will explain that Axiomatic design was applicable in the cases according to Suh [2], but could not be found in system architecture diagrams. We even encountered difficulties when trying to restructure the systems according to the Independence axiom.

Next in chapter 4 we will extend on the aspects that hindered direct application of Axiomatic design. The aspects show an interdependence between technical functionality and business terms. We call this interdependence Conceptual dependence conform McGinnes' definition [3]. The principle of Conceptual independence is explained in chapter 5. We propose a model for system architecture of IT systems with independent components in Chapter 5.

In chapter 6 Conceptual independence is positioned in the Design matrix of Axiomatic design. We conclude with stressing the importance of applying Conceptual independence when constructing IT systems based on Axiomatic design.

2 Description of case studies

The authors have studied three cases in health care in the Netherlands in the period 2012-2015 where existing systems needed to adapt to new health care requirements and processes. In the organizations a complex IT infrastructure functioned sufficiently and reliably for the current care organization. The board of directors in all three cases wanted to re-evaluate the IT systems in the light of new requirements. In all case studies we developed models of the IT architectures in collaboration with the IT architects in case 1, 2 and 3 or with the IT department in case 2. When compared to de facto IT architecture methodologies for change, the end goal of the IT architects was a Roadmap for adapting the existing IT system architecture to an IT architecture suitable for new requirements. This way of working resembles the TOGAF method in IT architecture practice [4].

2.1 Cases in health care

Two case studies were performed in hospitals, one hospital was still in the design phase and was planned to be a central node (with a physical building in Utrecht) in a network of care centers. The latter were departments in already existing hospitals. The other hospital was a regional medium sized hospital. The remaining case study was performed in a Home care institute that also provided facilities for intern patients and the elderly [5, 6].

The first case study took place from November 2012 until April 2013, the second from March 2013 to November 2014, the last from February 2015 until June 2015.

Examples of new requirements were:

• In all processes patients and their family are primary,
• Care processes and information must be organized around the patient and its family and not around medical specializations,
• All Information that is involved in the invoicing process must be integrated within the invoicing process as a whole,
• Achieve an increase in efficiency in planning of diagnosis and treatment with IT,
• Supporting IT for workflows of persons involved in planning of the ordering process of diagnosis and treatment.

2.2 Case study 1 National hospital for child oncology

The reason for developing a new system architecture was the establishment of a new national hospital for child oncology, that would integrate laboratory functions and treatment, integrate knowledge from research in this area and treatment and coordinate care centers across regions in the Netherlands. Information technology that existed in an academic hospital nearby would be reused where possible [7]. The IT architecture model, that was eventually selected by the IT architects can be seen as a simplified version of an enterprise architecture in which strategic goals were detailed in requirements for applications. In the Requirements analysis stage all strategic goals were specified into Services and those were specified in requirements for every one of the 90 applications.

2.3 Case study 2 Home care organization

The organization in this case functioned mainly as a care organization and provided home care and care for the elderly. It had 5000 employees and about 3000 volunteers at the time of study [8]. The IT architect and IT expert started with formulating strategic goals, because the organization was in the process of changing its health care vision and strategy. The ultimate goal however, was to judge if the current system was up to the required changes and could be maintained in its current form. The Board of directors was planning to update and replace parts of the system that were not functioning according to state of the art requirements.

2.4 Case study 3 Medium sized regional Hospital

In the third case study a regional hospital in the Netherlands is being studied [9]. The hospital is in the last stage of changing its paper-based operations to digital operations in the health care processes. The goal of the IT architects is to deliver a consistent vision of the changes in health care services as foreseen and the specified supporting IT functionality.

In the case studies the methods for changing existing systems were examined and notated. Practice showed that systems were difficult to change. We examined if in the existing systems the principles of Axiomatic design were applied.

3 Application of the Independence Axiom

Since in all three cases IT architects needed to adapt systems to new requirements, the lines of demarcation in the Information system architecture are crucial knowledge.

Our scope of Information system architecture is not confined to one Information system only. We have found that the IT systems in health care include Electronic Health records, in the Netherlands called Electronic Patient Dossier, systems for financial administration, systems for management, systems for special equipment such as MRI scan, systems for Enterprise resource management, systems for procurement.

3.1 Functional Requirements

In Information technology a distinction can be made between system functions and business functions. System functions are necessary in all organizations that need information for the working process such as storage of data, searching in collections of records, exchanging data between servers, exchanging data between servers and applications.

Business functions are functions that are useful in a specific organizations or industry, such as accessing a Patient Dossier, accessing and analyzing medical diagnosis, registration of a new patient, planning medical treatment in health care. In these functions specific health care terms are included in the system functions. Often the system functions can only be valuable for health care if information in information systems can be accessed with health care terms.

In papers of Suh regarding system design, he emphasizes the zigzagging over different domains when designing the definitive system [10]. In this paragraph I will explain how zigzagging over domains is applied in IT Requirements analysis. The Customer domain in IT consists of needs of the board of directors, but also of the medical specialists and other care professionals, such as nurses, and the information needs of patients. The

iterations of Customer domain to the Functional domain in the ideal case will lead to independent Functional requirements. Then it will be possible to link the FRs to the Physical domain. The preconditions for zigzagging from the Customer Domain to the Functional domain in Information system design are fulfilled. In case 1 the Requirements analysis was performed with 11 working groups of stakeholders. It resulted in 133 requirements with 793 Relations to Applications.

3.2 Decoupling Design parameters in IT

We have found that the Independence Axiom could have been applied in Information systems design in the same way as Axiomatic design prescribes. This is possible, because in realizing and programming of IT systems decoupling modules and functions by way of interfaces between different components in a system is good practice in software engineering. Patterns of decoupling system components are seen as qualitative high standards of software and are described in standard IT publications, such as in Larman [11], Buschman [12] and Bass [13]. Therefore we have concluded that Axiomatic design can be applied for defining independent or decoupled elements as Design parameters in IT systems in health care.

3.3 Conclusion of applicability of Axiomatic design

If we take the statements of IT engineering at face value, we can conclude that Axiomatic design can be achieved in designing and realizing IT systems. In the next chapter we will explain that demarcation lines in IT systems do not follow functional requirements.

4 Demarcation lines in IT systems in case studies

4.1 Case study 1 and 3: National hospital for child oncology and Regional hospital

Because we wanted to find the FRs of the existing systems in IT system architecture, first we have examined the most important FRs.

In case 1 in the hospital new requirements were formed because the vision of care as care for a patient and its family led to a changed vision of IT support. The IT support should adapt to organizing the systems around the patient and its family, instead of organizing the applications around medical specializations, its administration and registration needs. In case 3 strategic goals were specified in changing certain work flows for care processes. The new work flows should include communication and exchange of data with medical professionals from outside the hospital.

We have sought demarcation lines in IT systems for these FRs, and distinguished independent Runtime components in IT systems. The demarcation lines in existing systems in the hospitals were based on “applications” provided by suppliers for Information technology for health care.

We conform to the definition of application in this case study used by the IT architects in the study. An application can be defined in two ways:

1. A computer program that can be installed and run independently of other computer programs, and that delivers functionality defined by the supplier. The application can sometimes be coupled to medical equipment or can use open standards for communication. Features for network connections and communication over the network in proprietary standards and protocols are usually present.
2. A loosely coupled component in a computer program can also be an independent component, e.g. a module or functionality that can be turned off or on, depending on the possibilities that the supplier offers.

We have found in the cases that most of the functional requirements have been implemented in the applications in a way that is historically grown, and based on the knowledge that the suppliers had of the business processes of their customers, the health care organizations in the past.

Remarkably in both Hospitals was that a central position in the IT systems was reserved for the Electronic Patient Dossier. However the data belonging to the EPD was distributed among multiple sub applications that each have access to a part of the data of patients.

In both hospitals the IT system architecture consisted of applications that did not follow the distinctions between FRs.

4.2 Case study 2: Home care and cure

In the second case study we were interested in the divisions that IT technical experts were making when viewing the IT system architecture. We researched methods of change in IT system architecture in collaboration with technical IT architects.

An example of part of the IT system can be seen in Figure 1. In this figure the components in the system, do consist of applications (A, E and F), however these and the other systems are executing system technical functions, such as: storage of employee data and scheduling data, and exchange of data between one component and the other.

We can point out the following elements from part of the IT system architecture:

- A: The mobile application for input of employee data,
- A-B: A connection from the mobile device to the first proprietary database,
- B: The first database that communicates with the mobile device and stores the data in the IT systems,
- B-D: A connection from the first proprietary database to a second more general database,
- C: System that couples data to dashboards,
- D: The second database that collects all data from the first data storage database,
- D-E: Data output to combine data from data storage database to general information,

• E: The database that collects all general information, about history of illnesses, diseases, base tables. Also administration of invoices and finances. Indications for care in relation to insurances,
• E-F: Output of data from E to database F,
• F: The database that stores financial data for yearly reports.

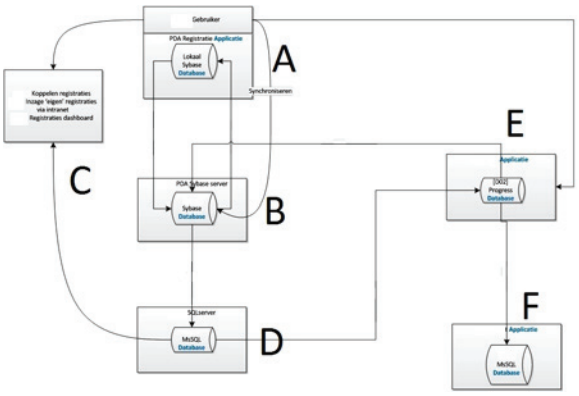

Fig. 1. Components in Home care institute.

The components are partly delivered by suppliers, partly these are custom made components. We add a third definition for an independent element in IT system architecture:

3. A computer program that can be installed and run independently of other computer programs, and that delivers more general system functionality. Examples are: data bases, input systems, file systems, hardware drivers, software components that enable network communication.

In this case the diagram focuses on system functions [14]. Applications are only named applications in this context when interaction with users is performed.

4.3 IT Actions for adapting systems

We will list the different IT actions in Table 1 that can change business and system functionality: Add a component, delete a component, change a component. However there is a fourth way of changing the IT systems, that is well known in software engineering as configuring systems with configuration files or in registry systems on the machine. Configuration are based on a structure that can be simple or more complex, they can even take the form of a conceptual model, with concepts that are structured and related. With changes in the content of the configurations (file) the behaviour of the Runtime component is changed without changing the component itself. Runtimes like these are called "computer language interpreters" or interpreters for scripting languages.

In all cases in Table 1, described above the changes must be foreseen. A component can only be added to a system when collaboration with other components is built in, in all system components that are concerned. A component can only be deleted when it does not contain functionality expected or needed by other components (dependencies). A component can only be changed, when changes do not affect communication with other components, or influence configurations of other components.

Table 1. Changing system by changing components and connections.

Behavior change	Structural change
Add component	Add communication line
Delete component	Delete communication line
Change component	Change communication line
Change configurations	Change structure of communication

A configuration of a component or a model based configuration file can only be changed in predefined ways, as the component "expects" configurations, and cannot cause unexpected or new behavior [15, 16], unless something goes wrong.

When the behaviour change concerns more than one component, communication between components must be changed. Connections can be added or deleted, thereby influencing the flow of information through the system. With the Independence axiom adding, replacing and deleting components or connections based on changing FRs will be feasible. We propose in the next chapters that also the option of "changing configurations" must be taken into account when designing Information systems.

4.4 Difficulties encountered when changing systems

In the cases that were studied we have found that the IT architects in comparison to experts from IT departments used different conceptual models for looking at the IT systems. IT (Enterprise) architects started with the strategic goals and vision on health care in mind and proceeded from there with defining business processes, such as in health care: Diagnose, Provide medical results, Propose treatment, Care plan, Medicine plan, Monitor and measure. These processes will be supported by IT services, and Functional requirements can be formulated as Functional requirements for these IT services [5]. The technical IT architects however, were looking at systems of system components and were interested in system failure and system functionality.

Despite the different views on the system, all cases showed that interdependency of conceptual data and system functionality obstructed required changes.

4.4.1 Patient data distributed and structure unknown

In the first case study, the changes required reorganization of applications, because the data for patients and their family was needed in a central place. IT architects found that data was distributed over different applications, and that the overall structure of patient data did not reside in one place, so to have access to the patient data structure from each application as needed. The data was structured in such a way as to accommodate the work flow and processes of medical specialists. Functional requirements were coupled to applications in a many to many relational structure. Therefore we conclude that the Independence axiom was not applied.

4.4.2 General system components adaptable

In the second case study the system consisted of system components and general system functions such as: storage, exchange of data and software for input devices. All components were dedicated to one system function, in which "business data and business information" were encapsulated.

The Functional requirements were not easy to locate in a system were information structures were hidden from view. Therefore we conclude that the Independence axiom had not been applied.

4.4.3 Work flow data distributed and work flow steps in applications are fixed

In the third case study in the Regional Hospital the IT architects sought to reorder the processes in which patients visited different departments in hospitals for medical examinations, medical results and diagnosis. The planning of these visits to different locations, the order in which the visits were planned was subject of study.

It was found, that although the application platform had been recommended as flexible, especially where workflows were concerned, the reorganization of workflows was not easy. The main cause of this, was the fact that documentation of the workflow formalization was difficult to understand. IT architects wanted to try out alternative workflows, but there were no tools supporting this. Functional requirements were not traceable in the previous workflows and we concluded that the Independence Axiom had not been applied. A clear distinction between Business concepts and technical implementation of workflows had not been made.

The problem of interdependencies of Conceptual models and technical logic in applications has been remarked and described as Conceptual dependence by McGinnes, we explain this in Chapter 5.

5 Conceptual independence

5.1 Conceptual dependencies

McGinnes describes how conceptual dependence can limit the changes one can make in independent elements in a system [3]. He describes Conceptual dependence as interdependency between the terms in the Business domain and the technical implementation of the application. In the previous paragraph we have shown that difficulties in adaptabilities in systems are the result of conceptual dependencies. We argue that a decomposition of functionality without separating the Business data from the system is insufficient.

Conceptual independence is based on separating the conceptual model from the technical application logic. A Conceptual model includes: all concepts, terms and instantiations from the business domain and their relations to each other, in a formalized model, it does not include behavior.

Conceptual independence is based on a meta model, in which the Conceptual model is filled in as data. For instance, the meta model consists of: Concepts, Relations, Attributes and Instances of concepts.
The characteristic of Conceptual independence is that the Conceptual model can change, without having to change the technical system that runs with the Concepts.
Because the Conceptual model is an instance of a meta model, it can be treated as data, we call it a soft model [3].

In a previous paper describing the pattern of Conceptual independence, we propose a complete division between the conceptual model where all business terms are collected in an Entity relationship-model (ER Model), as is customary in database design [14]. We then make a distinction in functions that are totally determined by the business and system functions that are general functions that can be applied also in other organizations. The latter need to be configured to this context/business. We propose the configuration of general functions instead of building functions anew for each organization, see model in Chapter 5.3. In 5.2 we give a recognizable example of Conceptual independence.

5.2 A Relational database management system as example of Conceptual independence

A well-known example of a Conceptual independent "application" can be found in a Relational Database Management System (RDBMS). This system can be deployed in any organization and filled with tables and table names specific to this organization. It can also store specific Business requirements in stored procedures and triggers that are fully determined by the business as concerned. These are functions that are programmed in a specific Data base language, dedicated for this purpose.
The DDL scripts in RDBMS can be seen as the Soft conceptual model, the database system as the Runtime conceptual model.

However, when applications are coded that access the database, the tables and names of tables are usually hard coded, and "expected" by the application, and therefore these applications do not conform to the principle of Conceptual independence.

5.3 A model for a complex IT system with Conceptual independence and Axiomatic design

If we combine Axiomatic design and Conceptual independence in an IT system we strive for replaceable components, for which each has the possibility to be adaptable to a specific Conceptual model. We outline this model in Figure 2 with Runtime components, each with its own functionality, such as: storage, exchange of data, access management, a rule engine. For specific functions, see Figure 3. These Runtime components all are based on the same meta model for conceptual models and can be configured with specific configuration languages.

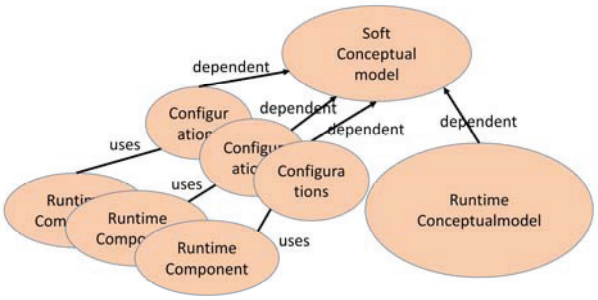

Fig. 2. Adaptable information system.

The Runtime components need a runtime Conceptual model such as a database system. A database system can be generated with a parser and generator system (not in diagram) without violating the Conceptual independence, see Figure 3.

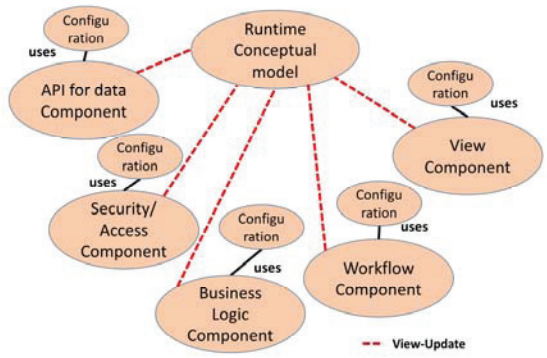

Fig. 3. Adaptable information system with system and business functions.

When the meta model, Soft conceptual model, the Runtime conceptual model and two Runtime components from Figure 3 are translated to a position in a Design matrix, we can map each independent Functional requirement to "one configuration" that adapts behaviour of the Runtime component. The configuration is then seen as a Design parameter. We will explain this in Chapter 6.

6. Adding Conceptual independence to the Design matrix of Axiomatic design

6.1 Soft Conceptual model and Runtime conceptual model added

We will describe an example of one general system function: "Show user interface for a specific Conceptual model" and one business function "Check consistency of data of one Conceptual model for an organization".

In Table 2 the Design matrix is represented. For each general functional requirement the Independence axiom is applied, which is dependent on the metamodel. The Runtime is "generated code" that means it will give access to information in the Soft conceptual model. The parser and generator system has been left out of the Design matrix for simplicity.

Table 2. Design matrix, example.

	Functional requirement	**Design parameter**									
		Meta-model	DP1	DP2	DP3	DP4	DP5	DP6	DP7	DP8	DP9	DP10
	Base meta model	x										
FR1	Store conceptual model	x	x									
FR2	Give runtime access to conceptual model	x	x	x								
FR3	Run rules	x			x							
FR4	Check Business Rule 1	x	x		x	x						
FR5	Check Business Rule 2	x	x		x		x					
FR6	Check Business Rule 3	x	x		x			x				
FR7	Present user interface at Runtime	x							x			
FR8	Present GUI specification 1	x	x						x	x		
FR9	Present GUI specification 2	x	x						x		x	
FR10	Present GUI specification 3	x	x						x			x

We could add a third Runtime component in the Design matrix for other services for the business, that can be decoupled from the system, but not from the Conceptual model. This component can be easily added in the same manner as the other Runtime components. One can check that Runtimes are reusable in other systems based on the same metamodel.

Table 3. Descriptions of DPs.

DP	**Description**
DP1	Conceptual Model in text
DP2	Runtime Conceptual model
DP3	Rule engine
DP4	Configuration 1 of Rule engine
DP5	Configuration 2 of Rule engine
DP6	Configuration 3 of Rule engine
DP7	user interface Runtime component
DP8	Configuration of user interface1
DP9	Configuration of user interface1
DP10	Configuration of user interface1

Descriptions of the DPs are represented in Table 3. Here every DP is described and the mapping of FRs to DPs can be traced forward and backwards.

Adding Conceptual independence to system design with Axiomatic design solves the interdependence of the technical application on business information and hard coded names and terms.

6.2 Design parameters and Process variables

In Axiomatic design another Design matrix can be constructed in which dependencies of Design parameters and Process variables are represented. Process variables refer to the way DPs are generated and therefore concern choices of resources. In IT we thus observe the choices of programming languages, human resources such as IT developers and specific tooling needed to deploy the system in a technical way [1]. In our example we have used a Metamodel for decoupling the Business terms from the technical implementation. It is disputable if this metamodel belongs to Design parameter or the Process variables. But, since it is essential for decoupling FRs we decide to explicitly include it in the first Design matrix in Table 2.

Table 4. Design matrix2, example.

DP	Description	Process Variable									
		PV1	PV2	PV3	PV4	PV5	PV6	PV7	PV8	PV9	PV10
DP1	Model in text	x									
DP2	Runtime Conceptual model		x								
DP3	Rule engine			x							
DP4	Configuration 1 of Rule engine				x	x					
DP5	Configuration 2 of Rule engine				x		x				
DP6	Configuration 3 of Rule engine				x			x			
DP7	User interface Runtime component								x		
DP8	Configuration of User interface1									x	x

Table 5. Design matrix2, example

PV	Description
PV1	Storage medium for Conceptual model
PV2	Database or Object base or other
PV3	Compiler 1
PV4	Syntax definition Rules
PV5	Rule1 in syntax
PV6	Rule2 in syntax
PV7	Rule3 in syntax
PV8	Compiler 2
PV9	Syntax definition UI
PV10	UI1 in syntax
PV11	UI2 in syntax
PV12	UI3 in syntax

However more can be said about the relation of DPs to PVs. In our example, we would need a medium for storage of the Conceptual model, compilers to generate Runtime components, a Runtime component that would provide access to the Conceptual model at runtime. For running the Rule engine and for presentation of the user interface, we need configurations in a syntax that is machine readable. Because we work with rules and screens that are independent of each other, independency of the configurations can be achieved. In Table 4 we show the PVs for the DPs in our example. The list of PVs is added in Table 5.

For readability, we have left out DP9 and DP 10.

In Table 5 we find different systems for storage, that are independent of each other, different syntaxes for the Runtime components and configurations in specific syntax for rules or UI screens.

7. Conclusion

Researchers have found in case studies that interdependence of Business terms and technical application logic is an important factor that made IT systems hard to change. We argue that FRs are independent, but that dependencies are implemented in systems in DPs by coupling Business terms and application behavior. Separating the business terms from the system components by means of a Metamodel, a Business data model and using configurations, gives designers the ability to define independent DPs. The PVs that have been described in this paper, demonstrate that the IT developers already have access to the tooling and resources to construct a system based on Axiomatic design.

8. Discussion

The ideas stated in this paper are difficult to accept by the IT architecture research community because in technical papers in software engineering the problem does not exist. We have demonstrated that the problem does exist in the case studies, where business functionality is primary.

In IT architecture research a separation of the Conceptual model from application logic is often seen as a theoretical construct that is not applicable in practice. In practice however, it can be observed that commercial companies are developing systems that generate conceptual independent applications. Three commercial businesses in the Netherlands present themselves as software developers for adaptable systems [17-19]. A first analysis of these platforms demonstrates that Conceptual independence has been applied, without the explicit reference to this principle. We are currently studying the extent to which they have also implemented the Independence axiom, and the Information axiom. Research with regard to these aspects is in progress. Another case study is in preparation in which a real-life system in health care will be developed based on these principles.

References

1. N.P. Suh, *Axiomatic design theory for systems.* Research in engineering design **10**, 4, 189-209 (1998)
2. N.P. Suh, S.H. Do, *Axiomatic design of software systems*, CIRP Annals-Manufacturing Technology. **49**, 1, 95-100 (2000)
3. S. McGinnes, E. Kapros, *Conceptual independence: A design principle for the construction of adaptive information systems*, Information Systems **47**, 33-50 (2015)
4. TheOpenGroup 2011. TOGAF Version 9.1 Evaluation copy. 9087536798, The Open Group.
5. D. Tarenskeen, S. Joosten, R. Bakker, *Using Ampersand in IT architecture,* in Proceedings of the CONFENIS-2013 : 7th International Conference on Research and Practical Issues of Enterprise

Information Systems, Prague, Czech Republic (TRAUNER Verlag, 201-209, 2013)

6. D. Tarenskeen, R. Bakker, Joosten, 2015. *Applying the V Model and Axiomatic design in the Domain of IT Architecture Practice.* Procedia CIRP. 34, 262-267 (2015)
7. UMCUtrecht 2012. Directieverslag 2012. http://www.umcutrecht.nl/nl/Over-Ons/Wat-we-doen/Jaarverslag-UMC-Utrecht, Accessed: 12-04-2017.
8. Carintreggeland 2014. Bestuursverslag 2013 Stichting Carint Reggeland Groep http://www.carintreggeland.nl/dbimages/Bijlagen/Bestuursverslag%202013%20Carintreggeland.pdf, Accessed: 4 -04-2017.
9. Interconfessionele_Stichting_Gezondheidszorg_Rivierenland, Culemborg', B.p.t. and Rivierenland', p.D.W.i.B.L.E.v.z. 2013. Jaardocument 2013. https://www.zrt.nl/upload/File/jaarverslagen/JAARDOCUMENT2013.pdf, Accessed: 4-4-2017.
10. N.P. Suh, *Axiomatic design: Advances and Applications* (Oxford University Press, New York Oxford, 2001)
11. C. Larman, *Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Interative Development* (Pearson Education India, 2012)
12. F. Buschmann, K. Henney, D.C. Schmidt, *Pattern-Oriented Software Architecture, Volume 4: A Pattern Language for Distributed Computing* (Wiley, Chichester, UK, 2007)
13. L. Bass, P. Clements, R. Kazman, R. *Software architecture in practice.* (Addison-Wesley Professional, Upper Saddle River, NJ, 2012)
14. D. Tarenskeen, *Conceptual Independence as an Architecture Pattern for Adaptable Systems,* in Proceedings of the VikingPLoP '16 (Leerdam, AA, Netherlands, 2016)
15. D. Garlan, R. Allen, J. Ockerbloom, *Architectural mismatch: Why reuse is so hard.* Ieee Software **12**, 6, 17-26 (1995)
16. D. Garlan, R. Allen, J. Ockerbloom, *Architectural mismatch: Why reuse is still so hard.* Ieee Software **26**, 4, 66-69 (2009)
17. Mendix 2016. Drive digital transformation & innovation with Mendix aPaaS, https://docs.mendix.com/, Accessed: 13-1-2017
18. Thinkwise 2016. Bedrijfssoftware - Thinkwise. https://www.thinkwisesoftware.com/nl/, Accessed: 13-1-2017
19. CGI-Group 2016. SMART Process Acceleration Development Environment. https://en.wikipedia.org/wiki/SMART_Process_Acceleration_Development_Environment, Accessed: 13-1-2017