TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ELIAS VAKKURI

DEVELOPING ENTERPRISE ARCHITECTURE WITH THE DESIGN STRUCTURE MATRIX

Master of Science Thesis

Examiners: Professor Saku Mäkinen and Associate Professor Marko Seppänen
Examiners and topic approved by the Faculty Council of the Faculty of Business and Built Environment on 9 October 2013.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tuotantotalouden koulutusohjelma
VAKKURI, ELIAS: Yritysarkkitehtuurin kehittäminen Design Structure Matrix -työkalulla
Diplomityö, 97 sivua, 3 liitesivua
Joulukuu 2013
Pääaine: Teollisuustalous
Tarkastajat: professori Saku Mäkinen ja yliopistotutkija Marko Seppänen
Avainsanat: Enterprise Architecture, tietojärjestelmäarkkitehtuuri, Design Structure Matrix, klusterointi,

Työssä tutkitaan, kuinka Design Structure Matrix (DSM) – matriisityökalua voidaan hyödyntää yrityksen tietojärjestelmäportfolion hallinnassa. Aihe liittyy laajempaan yritysarkkitehtuurin (Enterprise Architecture, EA) kokonaisuuteen, jonka tavoitteena on yrityksen liiketoiminnan ja tietojärjestelmien parempi yhteistoiminta. Yritysarkkitehtuurin viitekehykset eivät kuitenkaan yleensä tarjoa konkreettisia tapoja arkkitehtuurin kehitysehdotusten luomiseen. Tämä otettiin tutkimuksen kohteeksi.

Työ toteutettiin yksittäisenä tapaustutkimuksena, jonka tavoitteena oli lisätä kohdeyrityksen tietojärjestelmäarkkitehtuurin muutosjoustavuutta. Tarkasteltavat ohjelmat liittyvät yrityksen tilaus-toimitus –prosessiin. Yrityksessä haluttiin antaa eri toimipisteille vapaus päättää kuhunkin prosessin vaiheeseen liittyvistä tietojärjestelmistä itsenäisesti, mutta samalla varmistaa toimipisteiden tuottaman datan yhteensopivuus. Tältä työltä haluttiin tapa luoda kehitysehdotuksia, jotka täyttäisivät edellä mainitut tavoitteet.

Yhdistämällä tutkittavan yrityksen tavoitteet ja kirjallisuuskatsauksen tulokset tunnistettiin kaksi vaihtoehtoista suuntaa arkkitehtuurin kehittämiseen. Prosessin vaiheiden välinen tiedonsiirto voitaisiin toteuttaa joko suoraan vaiheelta toiselle, tai hyödyntäen koko tietojärjestelmän kattavaa väylärakennetta tiedon siirtämiseen. Työn käytännön osuus tarkastelee, miten nämä kehitysehdotukset voidaan luoda ja visualisoida DSM-työkalun avulla. Tämän lisäksi työssä tutkitaan myös DSM:n automaattista klusterointia. Tämän toivotaan luovan vaihtoehtoisia tapoja tietojärjestelmien ryhmittelyyn tilaus-toimitus –prosessin vaiheiden lisäksi.

Työn tärkein teoreettinen tuotos on tapa analysoida yritysarkkitehtuurin kehityksen aikana kerättyä dataa. Ehdotettu metodi on kehitetty yhdistämällä tietoa sekä tietojärjestelmistä että DSM:sta. Työn käytännöllinen tulos on konkreettinen työkalu, jolla voidaan luoda ja tutkia arkkitehtuurin kehitysehdotuksia.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Industrial Management
VAKKURI, ELIAS: Developing Enterprise Architecture with the Design Structure Matrix
Master of Science Thesis, 97 pages, 3 Appendix pages
December 2013
Major: Industrial Management
Examiner: Professor Saku Mäkinen and Associate Professor Marko Seppänen
Keywords: Enterprise Architecture, application architecture, Design Structure Matrix, clustering

This thesis examines how the Design Structure Matrix (DSM) tool could be used in managing an enterprise's information systems portfolio. The goal is related to a larger concept called Enterprise Architecture (EA), which seeks to better link together the business and the information systems in use. However, EA frameworks and previous literary works are often quite abstract in nature, offering guidelines on what data to collect but not what to do with the data. Thus, a practical method of analysis was needed.

The research was carried out as a single case study. At the case company, the primary goal for EA development was to increase the system's flexibility to change. The focus was on applications linked to the company's order-delivery process. The company wanted to allow each site freedom in choosing information systems for each phase of the process, but simultaneously ensure interoperability between sites. What was desired from this thesis was a way of generating development proposals for this type of architecture development, which could then be further refined.

Based on the case company's vision and a literature review, two distinct avenues were identified for developing the architecture. The communications could be organized so that information flows from module to module, or by defining a bus to move data between modules. The practical part of this thesis examines ways of generating these suggestions and visualizing them using the DSM. In addition, automatic clustering of the DSM was examined; this was seen as a way of generating clustering schemes alternative to those based on the process phases.

The main theoretical contribution of the study is a method for analysing data collected as part of EA work, created through combining theory on both information systems and the DSM. As a practical contribution, a concrete tool was developed during the study to facilitate the creation and analysis of the architectural suggestions.

PREFACE

This text marks the end of my journey as a university student. As is to be expected, the feelings are quite mixed, with the excitement of new horizons being contrasted with the anxiety of leaving a phase of life behind. Yet, in life as in many other matters, change is the only constant, a thing to be embraced, not avoided.

I would like to thank the following: Mr. Topi Putkonen for entrusting me with this project in the first place; and Mr. Topi Talvitie, Associate Professor Marko Seppänen and Professor Saku Mäkinen for their expert advice on matters both theoretical and practical. This thesis would not have been the same without their participation.

There are also a number of people who have my gratitude for getting me to this point in life. I extend my heartfelt thanks to my parents for their support and guidance; to my dear friends, both those I have met at TUT and those who have been at my side for longer; and to relatives who have been involved. Especially I would like to thank Tatu Marttila, Liina Sillanpää and Ritva Lamppu for helping me survive in Helsinki; and of course Jenni Ilomäki for her unending support during the process.

Onward to new challenges, in Helsinki November 20, 2013

Elias Vakkuri

TABLE OF CONTENTS

DEFINITIONS & ABBREVIATIONS

AA	Application architecture
DA	Data architecture
DSM	Design Structure Matrix
EA	Enterprise architecture
EIS	Enterprise information system
EAI	Enterprise application integration
SOA	Service-oriented architecture
TA	Technology architecture
TOGAF	The Open Group Architecture Framework
TOGAF ADM	TOGAF Architecture Development Method
VBA	Visual Basic for Applications

1 INTRODUCTION

This chapter discusses the background for the study, and the objectives and research questions to be examined. An outline of the rest of this thesis is also presented.

1.1 Background

1.1.1 Motivation for the study

IT environments in companies are becoming more and more complex (Schmidt and Buxmann, 2010, p. 168), and have been for decades. Already in 1985, (Porter and Millar, 1985) brought up the growing importance of information technology, while noting that it was not even then rare to have application portfolios with hundreds of systems running on and connecting through a plethora of different technologies. With the considerable evolution of computer hardware and software between then and now, the situation has not become any easier.

This becomes a problem, when in a quickly changing business environment, "enterprise systems need to be constantly and smoothly re-engineered" to match the market conditions (Chen et al., 2008, p. 647). This is hardly a trivial task, and the poor alignment between the company's business strategy and the information systems available is continuously a high concern among information system managers. (Luftman and Ben-Zvi, 2012; Luftman et al., 2009, 2005) The worst situation is when making changes to the application portfolio is so risky and expensive, that strategy and processes are built to cater to the information systems in use, and not the other way around.

Information systems management at the case company also faces the typical problems of an international, established organization, as presented by (Schmidt and Buxmann, 2010, p. 170): continuous modifications at distant sites, driven by local stakeholder concerns, implemented by various subcontractors while the systems are in use. All this adds to the complexity of the company's information systems environment. In addition, in enterprise systems as well as in single software products, (Lehman, 1996) Second Law of Software Evolution holds (Schmidt and Buxmann, 2010): the complexity of a software system increases if actions are not explicitly taken against it. This has been noted also at the case company: incremental evolution brings with it interconnections between applications that on the one hand make the overall system more efficient, but on the other hand often cause unexpected problems, when changes are made to applications. Thus, the flexibility of the application portfolio suffers even though efficiency increases. The information system becomes in this case "digital cement": when hardened due to the increase of interconnections, it cannot be changed without breaking the

system, and even worse, it also locks the surrounding organization in place. (Dreyfus, 2009, p. 5)

Other concerns include the costs and continuity of IT support. Local development and legacy systems that have come with mergers mean that some information systems may be in use in a very limited area, with only few people having intimate knowledge of the technology. However, these programs can be central to the operation of the whole system-of-systems. This represents an obvious personnel risk, as well as a cost source, if specialized support teams have to be maintained for systems with limited geographical use.

Enterprise Architecture (EA) has been proposed as one way to tackle these problem (Sowa and Zachman, 1992; The Open Group, 2011). There is a wide range of different EA frameworks and approaches that claim to represent a holistic view to the enterprise, with the aim of linking IT with the strategic goals of the organization. Despite few works presenting methods for quantitative analysis (Iacob and Jonkers, 2013, p. 249) or concrete evidence of benefits (Tamm et al., 2011), EA has spread widely in recent years (Schekkerman, 2005, p. 33).

The case company also has an EA program in progress, with the main focus being at the moment on the application architecture (AA), i.e. the specific collection of applications in use. A promising method for performing the AA analysis is the design structure matrix (DSM): a square matrix tool with the same elements in both rows and columns, used for mapping interactions between system components. The introduction of the tool is often credited to (Steward, 1981), who used the technique to analyze engineering design processes. Many other fields of application have been examined since (Browning, 2001; Eppinger and Browning, 2012).

The DSM seems like a good choice for two reasons. Firstly, several EA frameworks (DoD, 2010; The Open Group, 2011) propose using square matrices in modeling connections between system components. Secondly, many types of analysis approaches have been developed for the DSM. However, as discussed later in chapter 3, there are few literary works in which the DSM is used explicitly in an EA context (cf. Lagerström et al., 2013a, 2013b; Waldman and Sangal, 2007). In addition, none of these present practical guidelines for linking the DSM approaches to managerial implications, needed for practitioner acceptance. This is what this study aims to contribute.

1.1.2 Study focus

As a background, Figure 1-1 shows two example DSMs. The one on the left might show the situation after mapping interactions in a system, but before any analysis has been carried out. The interactions have been identified, but they do not form any logical groups, instead being spread out. The one on the right, on the other hand, shows interactions divided into loosely coupled groups. The aim has been identifying separate modules, one of the goals for which the DSM is well suited (Eppinger and Browning, 2012).

Figure 1-1: Two example DSMs

The more specific focus of the study is presented in Figure 1-2. EA development and planning often includes scenario analysis (Schmidt and Buxmann, 2010, p. 174). A simple scenario analysis method would include the phases in the figure: map the as-is architecture (1), choose criteria for developing prospective to-be architectures (2), make alternative plans (3 and 4), and then implement the best one (5).

This study focuses on steps 1 to 3 in the figure: applying theory to adapt DSM to the problem at hand, and to find criteria for good to-be architectures. The case company seeks open-minded to-be architecture proposals based on theory; these, while not necessarily implementable straight off, are hoped to give new perspectives on developing the architecture. Therefore, the to-be architecture is divided into theoretical and practical. There could be considerable differences between the theoretical and practical plans, which is portrayed in the difference between DSMs in phases 3 and 4.

The process is also meant to be repeatable: after changes have been implemented in

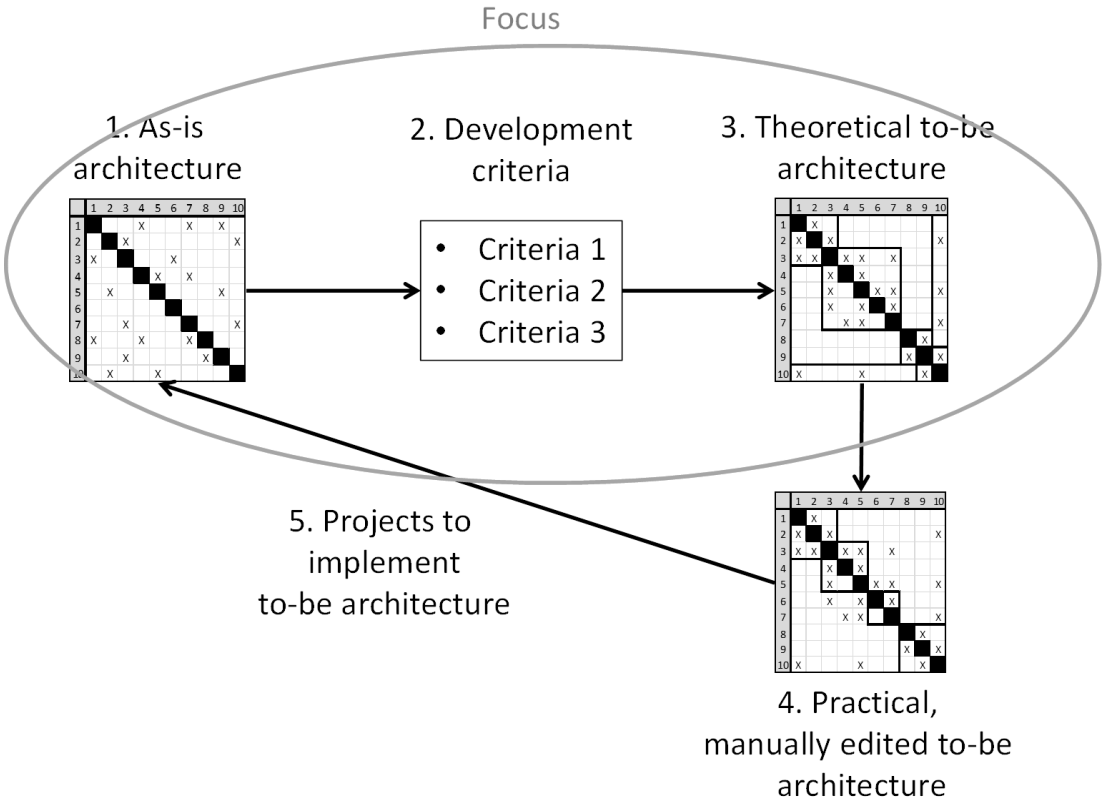


Figure 1-2: Research focus

phase 5, a new round of analysis begins at phase 1. However, forming the practical plans and implementing them in phases 4 and 5 are ruled outside the scope of this thesis, for two reasons. Firstly, the interest of the case company is mainly in phases 1-3. These contain reusable information that can in the best case be applied at different sites, forming a standardized approach for AA analysis. Phases 4 and 5 are much more dependent on the context, such as specific technological implementations. Secondly, the last two phases would require research on additional, extensive subjects, such as IT project management. This is not possible due to time and resource constraints.

1.2 Research problems & objectives

Based on the above, the main research problem of this thesis can be formulated as follows:

> *How can DSM be used to develop proposals for to-be application architecture as part of a company's information technology strategy?*

The question takes into account the specific interest in DSM as the approach; the tentative role of the architecture proposals as presented in Figure 1-2; and finally that the company's existing IT strategy will form the background for the development, and must be taken into account. This background could present itself e.g. in the criteria chosen for phase 2 of Figure 1-2.

However, the presented question is very general in nature, and so can be elaborated further. First, as the DSM is the main tool for analysis, any limitations it has will heavily affect the applicability of results. Therefore the question of highest importance is whether using the DSM requires making any assumptions or setting limitations. These must be made explicit.

The second question is what source data are needed in the analysis. This includes the elements and connections in the DSM. If the object of analysis is the AA, then an intuitive choice for the elements is the applications in use. However, this may not be the only opinion, and still leaves the question of the interconnections open. There may also be need for including other attributes in the elements than just names. These could the intended user group or the expected lifetime of an application, to name some examples.

Third, referring to Figure 1-2, there is the question of what criteria should be used to form suggestions for to-be architectures. What are objectives that are useful for the case company? Related to this is, of course, the question of what kinds of perspectives the DSM can provide. Finally, following from the third question, the criteria need defined measures. This allows an explicit comparison of as-is and planned to-be architectures, and as-is architectures at different times.

Thus, the more specific research questions are as follows:

1. *What assumptions and limitations have to be taken into account when using DSM?*

2. *What as-is source data are needed for forming a tentative to-be AA using the DSM?*
3. *What criteria (e.g. modularization) for developing an application architecture should be chosen, and why?*
4. *What are the metrics for the criteria?*

However, questions 2-4 are very much interlinked. On the one hand, if the DSM sets limits what source data can or should be used, then this limits what kinds of analyses and measurements can be performed, and therefore the needed source data should be defined first. Also, there could be practical limitations on how much and what types of data can be collected, which again affects the possible points of analysis. This view is reflected in the order of the research questions. On the other hand, if the limitations mentioned above are not critical, then the development criteria could be defined first and then collect the data relevant to these criteria. The most likely scenario is some combination of these two, and as such the respective order of questions 2-4 is not important.

Answers to the questions can be approached both from the general views presented in literature, and from the information needs of the case company. Previous research presents approaches that have been based on existing knowledge and often tested in practice, giving them credibility. However, to allow generalizability, they are always simplifications: applicable in many contexts while not exactly portraying any. Thus, it is unlikely that theory would present an approach with an "exact fit". As the case company is looking for practical gain out of this study, their particular needs will have to be reflected in the development criteria and measures, which in turn affect the needed source data.

As for the objectives of the study, the first one is of course to seek answers to the questions presented above. In addition, what the case company hopes for is a concrete tool for EA development. Ideally, this would be presented in the form of a step-by-step guide, so that the analysis can be repeatedly carried out in similar fashion. Thus, the objective of the study can be formulated as follows:

> *To offer the case company a step-by-step guide for using DSM in developing application architecture, taking into account the possible limitations of DSM and offering relevant development criteria based on literature and the needs of the company.*

When developing such a guide, one must again take into account not only the theoretical background, but also the constraints and interests of the business. As such, an approach proposed in the literature might not be suitable if it is too costly or time-consuming to implement, even if it would produce useful results.

1.3 Research methods

Background information and relevant theory were first collected on EA and DSM through a literature review. This was conducted in the latter half of 2013, mainly in June, July and August. The subjects of EA and DSM have different roles in this thesis, with DSM being the main focus and EA forming the "frame" in which DSM is applied. Thus, the literature reviews were not conducted in identical fashion. Specific details are given in chapters discussing these subjects.

The research itself is conducted as a case study. Although case studies often struggle to present generalizable theory, they are valuable in motivating and inspiring research questions and illustrating findings, if the conceptual contribution of the study is strong enough on its own (Siggelkow, 2007). The main contribution of this study will indeed be conceptual, aiming to propose a still lacking, concrete way of using DSM in AA analysis. This will be achieved by combining knowledge presented in literature and data from the case company for a comprehensive view based on both theory and practice. Thus, the case study will act as a way of forming ideas for the conceptual contribution and presenting the results achieved. In addition, case study as a method is closely linked to the reality of the case company, which eases the practical problem of researcher access. This is an important factor due to the limited resources available.

1.4 Study outline

The rest of the thesis proceeds as follows. Firstly, the results of a literature review are presented in chapters 2 and 3. Chapter 2 sets the background for the study by presenting information on EA: definitions for the term, common elements of different EA approaches, and the aims of EA. In addition, the current situation of the case company's EA work is presented. This forms a frame for chapter 3, where DSM is discussed. Answers to research questions based on literature are presented: what to analyze with the DSM, what data are needed, what assumptions are made.

Chapter 4 presents the approaches to be tested. In chapter 5, the case study, these are combined into a tool that can be used repeatedly. Finally, in chapter 6 the results and limitations of the study are discussed and possible roads for future studies are presented.

2 THE FRAME: ENTERPRISE ARCHITECTURE

In order to understand the frame of reference where DSM will subsequently be applied, a literature review was first conducted on enterprise architecture (EA). This chapter will present the findings, acting as background for chapter 3, which discusses using DSM in the EA frame.

2.1 Literature review method

As this section functions more as background, the aim was a broad rather than deep view into the field of EA. To this end, the first step was a search on the Scopus and Web of Knowledge (WoK) databases with the keyword "enterprise architecture". The most weight was given to articles with at least 50 citations, with special interest pointed to review articles.

Although articles with a high citation count have a proven track record, focusing just on them can omit more recent, yet important articles. Also, a keyword search can also leave out important works, if the right keywords are not found. Thus, the next step was more exploratory. This was done according to a method presented by (Webster and Watson, 2002, p. xvi). They propose three steps: a review of top journals in the field being studied, then a look backwards by exploring literature cited by works found in the first step, and finally a look forward into works citing the literature from the first step. The leading journals were chosen according to the suggestion by the Association of Information Systems (AIS Senior Scholars Consortium, 2011). These eight journals can be found in appendix 1.

As the author wanted to focus on more recent information, each journal was scanned from 2005 to 2013, providing that there was access through the Tampere University of Technology library. Although few works discussing EA were found, as noted also by (Tamm et al., 2011), this search uncovered interesting related concepts, such as enterprise application integration (EAI) (cf. Lam, 2005) and service-oriented architecture (SOA) (cf. Papazoglou and Heuvel, 2007). Afterwards, interesting works were looked for in works both cited by and citing literature found up to this point.

2.2 Definition

The ISO standard ISO/IEC 42010:2007 (ISO, 2007) defines "architecture" as follows:

> *The fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.*

Architecture therefore means both a "blueprint" for a system, as well as the general principles guiding the formation of the blueprint. In comparison with the traditional notion of architecture in the built environment, architecture can mean a design for a specific building, as well as the underlying principles of the design, such as in "gothic architecture". (Lankhorst et al., 2009, p. 2)

The division is further elaborated in The Open Group Architecture Framework (TOGAF), where architecture is defined in two parts. Architecture is firstly "a formal description of a system, or a detailed plan of the system at component level to guide its implementation", secondly "the structure of components, their inter-relationships, and the principles and guidelines governing their design and evolution over time". (The Open Group, 2011, p. 9) The second notion, architecture as guidelines and principles, is needed to communicate the formation of the system; in other words, "architecture is the set of descriptive representations that are required in order to create an object" (Zachman, n.d.). In order to be understood, a system must be depicted in terms that are familiar to all involved in the system.

In turn, an enterprise is defined as "any collection of organizations that has a common set of goals". Thus, depending on the level of analysis, an enterprise can mean a whole company, business or other organization, as per the traditional meaning of the word; in the case of a large corporation involved in multiple industries, it can mean a part of a company; and in the case of an extended enterprise, it can mean a company as well as its "partners, suppliers, and customers". (The Open Group, 2011, p. 5)

Combine the two definitions and you arrive at a definition for EA: it is a formal description of the structure of an organization with a common set of goals, and on the other hand a language for depicting said structure. Yet this leads to another question: if an enterprise is viewed as a system, what are its components? What makes up an enterprise?

This is of course dependent on what information is considered essential. However, in the EA literature, the main focus is the combination of business processes, the information systems that support them, and the infrastructure that the information systems run on. Presumably this has to do with the origin of the enterprise architecture concept. EA is seen to have evolved from the analysis of enterprise IT architectures, with (Zachman, 1987) being considered the seminal work in the field. (Richardson et al., 1990), on the other hand, is considered the first to have extended the focus to cover business concerns, to address the problems in integrating the aims of the business and the IT function. (Tamm et al., 2011, p. 142)

In keeping with this focus, EA can be defined as follows (Lankhorst et al., 2009, p. 3):

> *A coherent whole of principles, methods, and models that are used in the design and realization of an enterprise's organizational structure, business processes, information systems, and infrastructure.*

2.3 Contents of EA frameworks

There is a multitude of different frameworks available for EA (DoD, 2010; Sowa and Zachman, 1992; The Open Group, 2011). They often differ quite largely in contents: for example, TOGAF (The Open Group, 2011) focuses more on on the method and the meta-model for creating the architecture and less on the resulting architecture products (Ota and Gerz, 2011, p. 3), whereas the Zachman framework (Sowa and Zachman, 1992) does not provide any methodology for architecture creation at all, instead focusing on the schema of organizing information. Choosing one over another is not an easy task: (Johnson and Ekstedt, 2004) claim that "it is rarely evident when and why a particular model is to be preferred over others". The field of EA is still relatively young, compared architecture in for example building design, so a common, "time-tested" approach to EA has not emerged yet (Chen et al., 2008, p. 647). However, the different models contain similar elements. These are shown Figure 2-1, and discussed next.

Firstly, shown in section 1 in the figure, a fundamental idea of EA is to acknowledge the different stakeholder groups in the enterprise. A stakeholder is "an individual, team or organization (or classes thereof)" that has concerns related to "the outcome of the architecture" (The Open Group, 2011, p. 31). For example, a business manager may be interested in the business processes, how IS supports these processes and at what cost; a system administrator will focus on the maintainability and flexibility of the IS; and an end user wants the uninterrupted and quick operation of the systems. EA in the abstract sense aims at communicating "with and among stakeholders" about

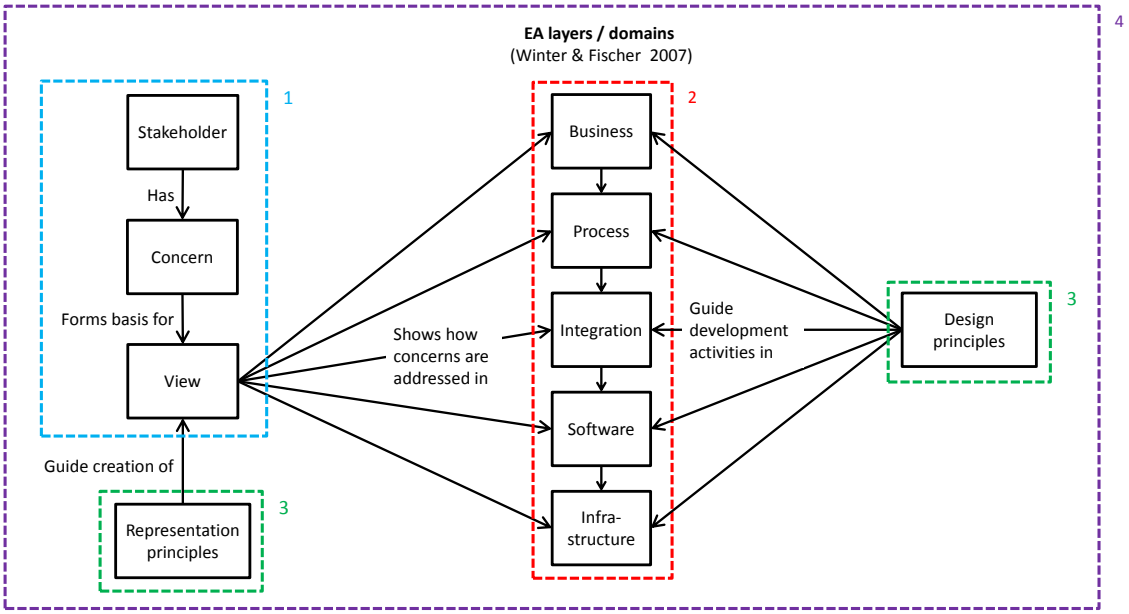


Figure 2-1: Common elements of EA frameworks

the concerns they have and how these will be taken into account (Chen et al., 2008, p. 649).

The concerns in turn form the basis for creating views to the EA. Due to the complexity of EA, it would not be feasible to try to address all concerns in one large architectural depiction. Thus, the views contain information relevant to a specific stakeholder concern. Architecture frameworks usually attempt to build the holistic view to EA by then combining these specific views (Ge et al., 2013, p. 365).

In addition to the stakeholder views, the EA is often divided also into different layers (Winter and Fischer, 2007) or domains (The Open Group, 2011). This is presented in section 2 in the figure. The layers in the figure are from (Winter and Fischer, 2007, p. 2), but many different domain specifications exist (Mikaelian et al., 2011, p. 458; Sowa and Zachman, 1992; The Open Group, 2011). Often these proceed from abstract to concrete, from strategic plans through business processes and information needs to software and technological infrastructure. Thus, the more abstract layers set restrictions and requirements for the more concrete layers, aiming to ensure that each layer is connected to the high-level goals of the enterprise. The views necessary for each layer guide the information content of the layers, to ensure that relevant stakeholder concerns are being addressed.

It must be noted, that there is disagreement on the terminology and the relationship between EA views and layers. Some frameworks subject the views to the layers (The Open Group, 2011), others place them on the same level (Sowa and Zachman, 1992), while yet other frameworks and authors make no difference between the two (DoD, 2010; Tang et al., 2004). This might have to do in part on the intended environment of the framework: DoDAF was designed primarily for use in the U.S.A. Department of Defense; therefore the information needs of the stakeholders can be clearly identified and there is no need for layers in addition to views. On the other hand, TOGAF is designed to be widely applicable, so it is prudent to present a list of layers relevant to most enterprises, while leaving the more specific definition of stakeholder views to the enterprise itself.

In the third section in the figure are the enterprise architecture principles. They are often presented as the most important part of an EA (Stelzer, 2009, p. 13), beginning already with (Richardson et al., 1990, p. 389). (Stelzer, 2009) divides these into design and representation principles. Representation principles concern the architecture as language: they contain “propositions for describing and modeling architecture” (Stelzer, 2009, p. 15), and therefore define how the architectural descriptions should be formulated. The chosen representation principles then should guide the notation in the views, so that common ways of communication are used in the enterprise. Examples of representation principles include understandability and completeness (Stelzer, 2009, p. 15),.

Design principles on the other hand concern the architecture as a blueprint, with rules and guidelines for the architecture itself. These include for example loose coupling and separation of concerns. (Stelzer, 2009) (Chen et al., 2008, p. 649) raise above all the

principle of fitness-for-purpose: EA should only be developed up to the point that the results remain useful.

Finally, a fourth element that EA frameworks usually contain is a toolset for managing the views and their change and development (Ota and Gerz, 2011, p. 1). The content of the views should be updated to match changes in the operating environment of the enterprise and technological development. In addition, stakeholder concerns and thus the views themselves also change (Pouloudi, 1999). Therefore, the architecture will need to be updated regularly to be of use (Schmidt and Buxmann, 2010, p. 170). To this end, TOGAF, for example, proposes the Architecture Development Method (ADM), a cyclical process from planning to implementation to measurement to planning again. The Zachman framework (Sowa and Zachman, 1992), on the other hand, contains only the set of views to depict the enterprise, and so it must be complemented with a method for updating the views.

2.4 Is EA valuable?

The end product of EA work should be a plan, a road map, for developing the relevant domains of the total architecture, based on a gap analysis of current and desired features, addressing stakeholder concerns. Ideally EA helps in both identifying necessary changes to the system as well as positioning these changes within the system, therefore "providing both stability and flexibility" (Lankhorst et al., 2009, p. 8). On the other hand, EA "marks the separation between what should not be tampered with and what can be filled in more freely". (Lankhorst et al., 2009, p. 4), acting as standards that ensure interoperability between applications, while leaving space for innovation (Boh and Yellin, 2006, p. 166). Thus, it provides a framework, a "skeleton", for subsequent filling with more detailed design work (Chen et al., 2008, p. 648).

Therefore, the expected value of EA comes from a unified approach. The lack of enterprise-wide guidance typically leads to local optimization and short-term solutions, even more so in a global environment, where the variety of stakeholder interests is manifold (Schmidt and Buxmann, 2010, p. 179). EA should enable reuse of "IT components and services", leading to obvious economies of scale, reduced complexity and a smaller pool of required IT skills. Another important goal is making possible for systems in different business units to function together. (Boh and Yellin, 2006, p. 166) This in turn is essential for cross-unit synergies that are required for quick responses to market changes (Tanriverdi, 2005).

Yet, realizing these benefits is far from easy. An EA plan, or even implementing EA itself, often call for large-scale changes to the organization, with the usual risks and challenges of change management.

Firstly, building and using EA is not only a technical challenge. Of course, a technically sound EA, that takes into account all relevant concerns, built with sound methods and resulting in a well-documented architecture plan, is a prerequisite for a successful EA initiative. Yet, even if technically sound, the plan must be both understood and ac-

cepted by all relevant stakeholders, not just the architects. Then, if the EA initiative has been performed in too much detail, it might turn out that the environment and the organization has changed too much in the meanwhile, rendering the EA plan obsolete. Yet another problem is the tradeoff between rigorous standards for the enterprise and the flexibility to innovate and change. The leadership qualities of the lead architect are of great importance here, and extensive communication and often corporate politicking are required to succeed. (Boster et al., 2000)

Secondly, EA has no value, if the plans and standards are not conformed to in the first place; the rules and guidelines set by EA must be enforced. Therefore, EA governance and the EA management function have a great impact on the success or failure of the EA initiative. This could present itself as, for example, reviewing proposed projects to ensure conformity. (Boh and Yellin, 2006; Schmidt and Buxmann, 2010)

Thirdly, quantitative analysis of EA scenarios is also difficult, and has not been widely discussed in literature (Iacob and Jonkers, 2013, p. 239). (Johnson and Ekstedt, 2004; Johnson et al., 2007a, 2007b) propose one approach, using influence diagrams in order to define concepts related to EA and dividing an EA concern into measurable subconcepts. However, even though the authors focus only on the concern of information security, and their examples have only a few components, the approach firstly requires defining a large number of constituent concepts, secondly creating extensive probability matrices for calculating how the defined concepts relate to each other. Both of these steps require a lot of human input in the form of either research into a subject, or expert opinion.

It is also worrying that there is little empirical evidence on whether and how EA actually adds value to an organization. (Tamm et al., 2011) As such, and with the challenges in quantitative analysis, a corporate-level initiative such as EA could be difficult to argument to individual business units; the costs start accumulating immediately while benefits are “long –term, global in scale, and difficult to measure” (Schmidt and Buxmann, 2010).

All in all, there are considerable challenges and open questions related to the use and value of EA in an organization. This is one of the reasons the case company is looking for robust analysis tools to aid in EA work. The next chapter discusses the DSM as an approach.

3 USING DSM IN THE EA FRAME

DSM usually stands for design structure matrix. However, the abbreviation is also used for many other but similar matrix-based design techniques, such as dependency source matrix, dependency map and N^2 matrix (Browning, 2001, p. 293). The term DSM will be used in this text. This chapter focuses on how DSM is used as a method in general and in the context of software.

3.1 Literature review method

Due to DSM being unknown to the author, the first step was to search for general information on the subject. The starting point for the search was (Eppinger and Browning, 2012): with a wide selection of examples from different industries, it gave a good overview of applying the DSM. The authors are also well-known in the field, so the representability of their selection of examples could be treated with confidence. The examples that they present were also highly cited in Scopus, WoK and Google Scholar. This approach gave insight into fields where DSM has been applied, and in what types of analyses. These are considered in chapter 3.2.

However, for specific information on using DSM in analyzing EA, a more specific search was needed. Thus, the next step was a search for literature on using DSM in enterprise architecture. The first search was made on Scopus and WoK using as keywords known terms for both enterprise architecture (enterprise architecture, application architecture, IT architecture, service-oriented architecture, information systems architecture, data architecture) (Bidan et al., 2012, p. 289) and for DSM (DSM, design structure matrix, dependency structure matrix, dependency structure method, dependency source matrix, dependency system model, deliverable source map, problem solving matrix, incidence matrix, N^2 matrix, interaction matrix, dependency map). After the initial search, more keywords were identified, and were subsequently added to the search. The total list of keywords used is shown in Table 3-1. The specific search strings used in this phase can be found in appendix 2.

However, there were limitations to what articles were considered useful at this stage. Firstly, articles related to how to use the DSM technically were not the goal of this search, as they had been considered in the previous step concerning DSM in general. These included e.g. literature on algorithms for modifying the DSM. Secondly, many articles were not suitable for the case company's needs. There were several reasons for this. Some articles were specifically related to a certain industry, and applying these to the case company's context would have been difficult. The case company also had already performed extensive data collection, and thus these data formed a frame in which

Table 3-1: Search keywords for DSM in EA setting

Subject	Enterprise architecture	DSM
Keywords	enterprise architecture	dsm
	application architecture	design structure matrix
	IT architecture	dependency structure matrix
	service-oriented architecture	dependency structure method
	information systems architecture	dependency source matrix
	data architecture	dependency system model
	technology architecture	deliverable source map
	system-of-systems	problem solving matrix
		incidence matrix
		N^2 matrix
		N squared matrix
		interaction matrix
		dependency map

the analysis methods would have to fit; radical changes to the source data were not possible. This ruled out several articles, which presented concepts building on certain data.

In the end, three articles filling the requirements were found, sadly none of them peer-reviewed: (Brøndum and Zhu, 2010; Simpson and Simpson, 2009; Waldman and Sangal, 2007). A later exploratory search uncovered two additional working papers, namely (Lagerström et al., 2013a, 2013b).

However, with only five pieces, the theory used in this thesis could not be based on these sources alone. Thus, the requirements were relaxed to an extent, in order to find usable analogies between existing literature and the case company's situation. This brought back to consideration some articles that had been previously excluded.

In addition, the general research into DSM had revealed that it has been widely applied in the analysis of software in general. This felt like a logical match. After all, there are obvious similarities between a single software product and AA, as AA is in effect a system of software systems. Thus, in theory, each element of an AA DSM could be "exploded" to show in greater detail the exact connections within a software system. However, differences were to be expected as well, due to e.g. the lower granularity of an AA DSM. Also, it was not clear, if a single software system and a system of systems can be analyzed in the same way. These considerations were taken into account in assessing the literature's applicability. They will also be discussed in the applied section of this thesis, from chapter 5 onwards.

Looking for additional literature, the sample of literature was widened through snowball sampling, adding items that appeared often in other literature or seemed interesting based on the citations. Of course, this made the research more exploratory than systematic, but this was necessary due to the field being unknown to the author.

The rest of chapter 3 will present information gathered on the DSM. First, general guidelines on using DSM are discussed in chapter 3.2. Then, more specific results for

using DSM in EA and software are presented. Chapter 3.3 considers interactions, the marks in the DSM, and chapter 3.4 the types of analysis that the DSM should then be subjected to. This will give theoretical tools for the research presented in later chapters.

3.2 DSM in general

3.2.1 Basic information on DSM

In all simplicity, the DSM is a "square matrix, with the rows and columns identically labeled and ordered, and where the off –diagonal elements indicate relationships between the on-diagonal elements" (Eppinger and Browning, 2012, p. 6). In this thesis, the elements in the rows and columns will be called "elements", and the elements populating the matrix will be called "marks". The elements represent components of the system being analyzed, whereas the marks represent "relationships" between the components.

The description above is intentionally highly general; that is part of the usefulness of the DSM. Firstly, when defining "component", the most important requirement is that what the components represent is suitable for the domain being analyzed and the desired granularity of the analysis. The dictionary definition is representative in this context: a component is a "constituent part" of a system (Merriam-Webster, n.d.). As such, the DSM as a method sets no restrictions on what systems and what components can be analyzed; what is important is that modeling the system yields useful information. In addition, every complex system consists of a nested hierarchy of other subsystems (Simon, 1962). Thus, every component could theoretically be "exploded" to show its constituent components, or a group of components could be "consolidated" into a higher-level component or module. What limits this zooming ability is the usefulness of the results and the availability of information on the inner structure of the components.

Regarding what the elements usually represent, (Browning, 2001, p. 293) proposes a two-level hierarchy for DSM classification. The first distinction is between static and dynamic DSMs: in a static DSM, all the elements in the DSM exist simultaneously, whereas a dynamic DSM also has a time dimension, i.e., an element further down the DSM is worked on later than an element which is higher.

On the second level, static DSMs are further divided into component-based or architecture DSMs and team-based or organization DSMs (Browning, 2001, p. 293). As the names suggest, an architecture DSM is often used to analyze the architecture of products, either concrete (e.g. Pimmler and Eppinger, 1994) or abstract (e.g. MacCormack et al., 2006 in software). Organization DSMs are used in depicting the structure of an organization, for example in order to identify logical teams based on the frequency and importance of interactions (Eppinger and Browning, 2012, p. 84).

Dynamic DSMs can be further divided into activity-based or schedule DSMs and parameter-based DSMs (Browning, 2001, p. 6); however, (Eppinger and Browning, 2012, p. 11) argue that the two are very similar in use, and combine both under the category of "process architecture DSM". As said, these types of DSMs have a temporal di-

mension, and as such they are used in mapping for example processes and projects, with the DSM elements representing activities (Eppinger and Browning, 2012, p. 131).

Secondly, "relationship" is defined with similar abstraction; the relationships can depict anything relevant to the domain being studied. Often, the word "interaction" is used instead of "relationship". The marks populating the matrix can depict different types of interactions depending on the context: transfer of material, energy or information (e.g. Pimmler and Eppinger, 1994); communication between persons or teams (Sosa et al., 2004, 2003); or needed inputs from one process phase to another (Danilovic and Browning, 2007). In addition, the marks can be either binary – i.e., there is or is not an interaction between two elements – or numerical – i.e., with a value giving additional information on the interaction.

Moreover, two conventions exist regarding the direction of the interactions: the inputs for an element can be marked either in the element's row or column, and conversely the input in the other. The choice is not critical, as either one can be transformed into the other by transposing the matrix. What is important is that the choice is made clear. (Eppinger and Browning, 2012, pp. 4–5) **For later analyses, this thesis adopts the convention of outputs in rows and inputs in columns**.

3.2.2 Modifying a DSM

It is stated that already the process of forming a DSM can have benefits, as it increases knowledge of the system being examined (Eppinger and Browning, 2012). However, methods for modifying the DSM have also been developed; often this means rearranging the elements in a DSM so that they form a better architecture by some metric. The aims for modifying are different for static and dynamic DSMs. Static DSMs are usually "clustered", while dynamic DSMs are "sequenced". These are examined next.

Clustering aims at arranging the elements into clusters: sequential sets of DSM elements grouped to "achieve efficiencies through common membership in the cluster". Examples of grounds for clustering include being "produced by a common supplier, sharing multiple interfaces, or having complex interactions". (Eppinger and Browning, 2012, p. 24) Thus, clustering a DSM means reordering the elements, or the rows and columns, so that the chosen objective is achieved.

When clustering, it does not matter on which side of the diagonal the marks are, as there is no time element involved. In sequencing, on the other hand, marks below the diagonal are undesirable as they represent feedback, or information which is needed before it is available (note that this is dependent on the chosen convention on the direction of the interactions). Sequencing therefore has two aims in arranging the elements: firstly, to keep the amount of feedback marks to a minimum; secondly, two keep the required feedback loops as short as possible. (Eppinger and Browning, 2012, pp. 141–143)

It must be noted that limiting clustering only to static DSMs and sequencing to dynamic DSMs is an unnecessarily black-and-white approach, and (Browning, 2001) presents several ideas for widening this view. In the software domain, both clustering

(MacCormack et al., 2006) and sequencing (Sangal et al., 2005; Sosa et al., 2007a) have been applied to essentially identical DSMs, thus pointing out that one should not limit oneself to only one type of analysis.

Clustering and sequencing can give insight into the architecture, but they only result in permutations of the same matrix. An additional method is tearing, which means a temporary removal of marks (Steward, 1981). Tearing allows more flexibility in the analysis, as it truly changes the DSM. Of course, this kind of modification can be extended to adding marks and adding, removing or combining elements, but for some reason these have not been given special names.

Using the general methods above and possibly other, case-specific methods, DSM analysis is then usually carried out by exploring several possible scenarios and weighing "the pros and cons" of each. (Eppinger and Browning, 2012, p. 89) What these pros and cons are depends on the domain; e.g. in organization DSMs cons could be "physical or political constraints on the size and composition of groups, such as the size and/or location of a facility and established reporting relationships" (Eppinger and Browning, 2012).

Of course, what the pros and cons are is not a trivial question, and depends again on the circumstances where the DSM is being deployed. Many kinds of metrics have been developed for e.g. measuring the modularity of the analyzed system; these are given a more in-depth appraisal in chapter 3.4.3. By comparing the metric values between the current and proposed DSMs, the system architect can gain some insight into how the changes will affect the system.

3.2.3 Obstacles and challenges in using DSM

As said, the DSM is a very general type of tool, as are all matrices. As such, the DSM in itself does not set any limits or assumptions on its use. More important is that the information put into the DSM is as accurate as possible, and represents the problem at hand accurately. This is where the problems lie.

Firstly, collecting data for the DSM relies often on human input: information on elements and interactions are collected from the tacit knowledge of experts using e.g. interviews and surveys. Especially when analyzing a large system, the error-prone nature of the human mind can introduce flaws into the DSM. Sometimes researchers can omit by accident persons, who would have valuable inputs. As such, "the quality of the DSM depends heavily on who was invited to provide inputs to the DSM". (Eppinger and Browning, 2012)

There are ways to tackle this. The data should be gathered in the directions of both the inputs and the outputs; in other words, the experts should all be asked, which components provide inputs to their components, as well as who they provide inputs to. This gives a two-way view into the architecture, and possible inconsistencies can be discussed.

In addition, recognizing and gaining access to everyone that could have valuable input is at the very least time-consuming. Some go as far to give quantitative estimates:

"the effort for manually creating a product architecture DSM" has been estimated as $T = 0.02 * N^2$, "where T is the number of person hours and N is the number of components (parts or subsystems) represented in the DSM model". Specific values could of course be case-specific, but it seems generalizable that the time consumed scales strongly with the size of the DSM. (Eppinger and Browning, 2012, p. 44)

It helps if the information can be gathered from explicit knowledge, e.g. design documents. Additionally, especially in the software domain, automatic gathering of data is often a viable possibility (e.g. Sangal et al., 2005). This allows the data gathering for even systems of thousands of elements to be completed in a reasonable time.

Secondly, the DSM is not the best tool for every situation. The DSM has close links to graph theory, as any DSM can be presented as a directed graph or digraph, with the elements of representing vertices and the marks representing edges. The DSM is claimed to have many benefits using a digraph to visualize the same information, such as conciseness, intuitiveness and a wide selection of tools for analysis. (Eppinger and Browning, 2012, pp. 4, 12) On the other hand, (Sharman and Yassine, 2004) show cases where a digraph would present the system structure unambiguously, whereas a DSM, and other 2-D representations for that matter, give an unnecessarily complex view. The problem is presented in Figure 3-1.

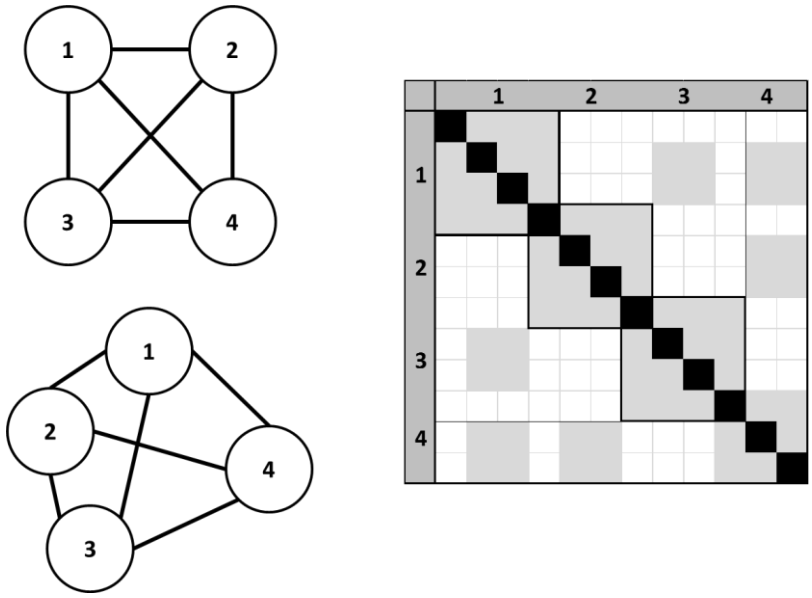


Figure 3-1: Comparison of graphs and DSM (adapted from Sharman and Yassine, 2004, p. 46)

In the figure, groups of components 1, 2, 3 and 4 are equally connected to each other, as presented in the graphs. However, this is not apparent from the DSM. Instead, cluster 4 appears as a "bus" component, whilst clusters 1, 2, and 3 can be considered as modules, "with some semirandom crosslinking" between clusters 1 and 3. However, any of the clusters 1 to 4 could have been set as the bus. Thus, the real structure of the system cannot be deduced unambiguously from the DSM, which can have consequences

when searching for an optimal layout. Therefore, both representations have their merits. (Sharman and Yassine, 2004, pp. 46–47)

Thirdly, the choice of the abstraction level has a great impact on the usefulness of the DSM. If the level of abstraction is too low, too granular, then the intuitiveness of the DSM suffers as there is too much information. On the other hand, a too high abstraction level could leave out important details on sub-system linkages. This can be helped by constructing a multi-level DSM, each line on a high-level DSM with its own, more granular DSM. Thus the level of abstraction can be changed as needed (Lindemann, 2009) Needless to say, this multiplies the amount of information needed, and the problems that rise with that fact. However, if e.g. the information collection can be done automatically, then this approach is very viable.

3.3 Interactions

An essential step in constructing a DSM is to decide what data will represent the interactions between the elements. However, the choice is of course largely dependent on the frame where the DSM will be applied. The important questions to ask are, what constitutes an interaction, and what does the interaction imply for the design.

As previously explained, in analyzing physical products the marks can represent e.g. requirements for elements to be situated closely together, or to be able to pass materials or energy between one another, or for team members to communicate effectively. In short, the connections have implications in the physical realm, as e.g. fasteners or cables between two elements, or decisions to co-locate certain personnel. They will also be important when seeking a better architecture, because certain set-ups could be costly or even physically impossible to implement.

On the other hand, software, and by extension enterprise architecture consists solely of information. Therefore using a similar taxonomy to physical products has no value, as all the connections will be information, and further there will be no physical limitations to implementing the architectures (Baldwin et al., 2013, p. 29). Thus, no architecture will be strictly impossible. Therefore, it is to be expected that interactions in the software domain differ from those used in the physical domain.

Based on the literature, authors use interactions of varying levels of abstraction in the interactions in software DSMs. Firstly, the interactions can represent specific syntactic connections. Examples include global variables (Schach et al., 2002) and function calls (Baldwin et al., 2013; Banker and Slaughter, 2000; MacCormack et al., 2012). These works focus on just one type of interaction, which is seen to be of most importance.

A higher level of abstraction is presented in works based on (Sangal et al., 2005), in e.g. (Hinsman et al., 2009; Langmead, 2007; Roosmalen, 2007). They use the marks to represent explicit references in code to “syntactic elements” of a software system, thus any type of syntactic connection is marked. (Waldman and Sangal, 2007) expand this approach to analyzing enterprise architectures, by adding to the DSM connections to

e.g. databases and business processes. An even more abstract approach is to have the marks mean simply a design dependency: if A depends on B and B changes, then A might have to change as well (Lopes and Bajracharya, 2005; Sullivan et al., 2001). This type of approach would seem to lend itself well to developing a new architecture, as it is highly analogous with a task-based DSM: the dependencies are information which must be known before designing the dependent component.

All the previous examples either require access to the source code or to personnel who are familiar with the design of the software system. The other option is that the dependencies are implied by some components often being changed together (Cataldo et al., 2006; Eick et al., 2001). This type of interaction is useful, as it gives an unambiguous probability for changes propagating to other components. However, studying these types of connections requires access to maintenance data, which are not always available and are not gathered in the same way in different projects, and so might not be comparable (MacCormack et al., 2012, p. 1312).

What is common to all these streams of literature is that the marks in the DSM should represent the impact that changing one component has on another, as presented in e.g. (Sharman and Yassine, 2004). Thus, the different interaction types presented in the literature can summed as dependency: if a change is made in on component, another might have to change as well. Either the dependencies are found out statistically (statistical dependency), defined as part of the design of the program (designed dependency), or approximated by using a proxy value (specific / any syntactic dependency). This is summed in Figure 3-2.

However, one still has to decide between using binary and numerical interactions. Sadly, the authors don't always document their approach very clearly. For example, the works by MacCormack et al. state that the interaction they use is the function call. Yet

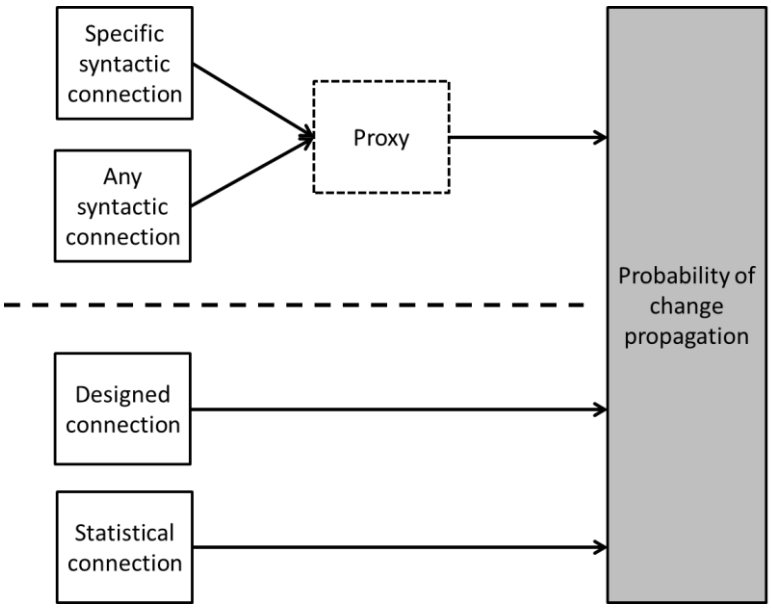


Figure 3-2: Summary of interaction types

they don't say if the marks are binary or numerical. In other words, is one function is enough to put a mark between two elements, or is the strength of the interaction measured by e.g. the number of the function calls? A mark in a binary DSM basically means a 100% probability of a redesign, whereas a numerical DSM, if normalized, can represent probabilities anywhere between 0% and 100%.

One possible approach would be to apply ratings for different software interactions, in accordance with the level of the coupling. (Fenton and Melton, 1990) present a widely used measure (Lagerström et al., 2013a, p. 4) for the complexity of software coupling, based on a six-level ranking of interaction complexity. Their metric equation is as follows:

$$M(x, y) = i + \frac{n}{n+1}$$

Where *M* is the complexity of the interaction between elements *x* and *y*, *i* is the measure of the most complex coupling type between the two elements, and *n* is the number of interconnections between the elements.(Offutt et al., 1993) later refined this metric to the following:

$$M(x, y) = i + \frac{n}{n+1} - \frac{1}{2} = i + \frac{n-1}{2(n+1)}$$

in order to bring the metric value closer to the value of *i* rather than *i+1*. It is easy to see that metrics such as these are easy to implement in matrix-based analysis, provided information on the connection types is available. However, despite their popularity, (Xia, 2000) argues that the ranking used in the metrics above does not reflect reality. Instead, they propose a measure based on three attributes: complexity of data, program control, and number of connections. They claim that these attributes capture the ranking by (Fenton and Melton, 1990) and are additionally more flexible in reflecting real-life situations. In a similar stream, (Brøndum and Zhu, 2010) presents a taxonomy of five different interaction types: 'relates to', 'depends on', 'constrains', 'refers to' and 'connects with'. The fifth interaction itself links with the sizeable literature concerning software connector types (Mehta et al., 2000). The interactions are presented an "annotated DSM" (Brøndum and Zhu, 2010), with each mark containing references to all the types of interactions existing between the two elements.

The examples above present interesting possibilities for quantitative analysis in a DSM. Using some of the metrics presented allows estimating the connection strength numerically. However, it has to be noted that most of the interactions presented above have been used in analyzing a single software system, not the enterprise architecture. It is not clear if the same tools are useful in both domains; at the least, information collection in EA domain is much more reliant on manual work, e.g. surveys and interviews (Lagerström et al., 2013b, p. 17). This means that, due to resource constraints, the information needed for certain metrics might not be able to be collected in a cost efficient fashion.

The literature on DSM in EA domain gives few answers to the binary-numerical question. Even though (Lagerström et al., 2013a) cite various complexity metrics, they eventually only present their marks as "dependencies"; although, as they base their research approach on (Baldwin et al., 2013), it is probable that they use the same function call proxy. Yet they don't give reasons for this choice. (Lagerström et al., 2013b) open their approach a little more, and define the different types of interactions as "communicates with", "runs on", "is instantiated by" and "uses". Again, no reasoning for this taxonomy is given. (Waldman and Sangal, 2007) follow the convention set by (Sangal et al., 2005) that any syntactic connection causes a dependency, but no reasoning is given in this case either, why the chosen type of interaction would be important.

All in all, the literature seems to agree that the interaction type chosen should represent the probability of change propagation across components. However, what kind of interaction best captures this view is not as clear. The choice is also contingent on the ease of data collection and, intuitively, on the variety of the interactions present in the EA.

3.4 Objects of analysis

After deciding on what the elements and interactions represent, the initial DSM can be formed. The next logical step is to analyze the DSM and look for opportunities for improvement. This chapter considers what information can be gained from DSM analysis, based on methods used in literature.

3.4.1 Adherence to design rules

In order to keep the design of a system manageable, rules have to be developed. These design rules are high-level decisions that define how elements in a system are developed. By defining design rules, otherwise coupled design decisions can be separated, and afterwards components and modules can be developed independently of others. This in turn increases the value of the design by creating real options: each component can be substituted with a more advanced one regardless of other components, thus representing an option for change that can be exercised if seen as beneficial. (Baldwin and Clark, 2000)

As such, in the software domain design rules can represent e.g. common interfaces to which components must adhere (Lamantia et al., 2007). Several works in analyzing software with the DSM concentrate on using the DSM as an alternative to other diagrams often used in software design, such as UML (Avritzer et al., 2010; Lopes and Bajracharya, 2005; Sullivan et al., 2001). In these works, the design rules are represented as a separate module in the DSM, consistently with the approach of (Baldwin and Clark, 2000). The rules can separate the design of the software modules firstly from each other, secondly from external sources of change, such as the user, the computer environment or third-party elements in the system.

(Sangal et al., 2005) and derivative works take a different stance. In their approach, design rules are not portrayed as elements in the DSM, but as rules stored in a separate system, such as "Element A cannot use element B". These rules are then used to identify interactions between components that violate the rules, which can then be highlighted in the DSM. A primary aim is to maintain a layered nature in the software system, in other words that elements can only use elements below them in a hierarchy. This is done in order to eliminate cyclical dependencies, which can cause the propagation of changes.

There is also an interesting logical connection between design rules and bus components or modules, which are common in many real-life complex systems (Sharman and Yassine, 2004; Sosa et al., 2003). An integrative component has connections with a large number of other components, and as such it can represent an interface between components or modules. This is especially true, if other elements are proportionally more dependent on the integrative component than vice versa; in this case it would truly be analogous with a design rule, as other components would have to be compliant with the bus component, but the bus would not be affected by changes in other components. Of course, the bus component is not an interface in sense of e.g. an industry standard; each component connecting to the bus could do it with a technically different interface. However, the bus can be made a stable part of the architecture, allowing modules connecting to the bus to be developed independently.

Yet another connection is to the concept of enterprise application integration (EAI) (Bidan et al., 2012; Lam, 2005). EAI aims at creating channels of communication for otherwise separated "islands of automation": groups of applications with little or no connectivity to other information systems in use. One of the approaches is defining and "enterprise bus", a backbone for the EIS to which applications can connect and by which they can communicate using explicit interfaces. This is again logically very similar to design rules, as an enterprise bus decouples the internal implementation of an application from its external interface. In addition, an enterprise bus is naturally a bus component in the system, with the attributes discussed.

All of the definitions above can be of use when analyzing and developing the application architecture. The design rules, as defined by the system architects, can be shown as separate elements or as visual instructions on acceptable dependencies. Also, if some components can be identified as buses, they can be made into a sort of design rule, with their interfaces guiding the development of other components. This type of analysis can help to bring structure to the AA.

3.4.2 Core and peripheral components

(Tushman and Murmann, 1998) divide components of complex systems to two categories: core and peripheral. They base their division on the degree of coupling: core components are those which are tightly coupled to other components, meaning that there are many connections between them and other components. Conversely peripheral components are loosely coupled, having fewer connections. (Murmann and Frenken, 2006, pp.

940–943) extend the analysis to the components' *pleiotropy*, a concept loaned from biology, meaning the "number of traits affected by a particular gene". Thus, in systems analysis, pleiotropy is defined as the number of service characteristics affected by a change in a component. High-pleiotropy components, or core components, are linked to many service characteristics, whereas peripheral components have low pleiotropy and affect only few characteristics.

Consistent use of certain high-pleiotropy components can shape entire industries, with the most common combination becoming the dominant design. Stable technological choices in the core components, the dominant designs, function as interfaces, which makes possible the incremental development of other components. On the other hand, changes to core components can not only lead to changes in many other components, but also in the service characteristics visible to the user of the system. The extensive influence core components have cause changing them to be highly risk-ridden. (Murmann and Frenken, 2006; Tushman and Murmann, 1998)

The classification of components as core or peripheral connects with the literature on component modularity (MacCormack et al., 2010, 2007; Sosa et al., 2007b). This is different from system-level modularity, which is discussed in chapter 3.4.3. These works argue that modularity is a component-level property, rather than system level. The more loosely coupled a component is, the higher its modularity, with tightly coupled components classified as integrative. Studies conducted in the software domain give evidence to the large influence of tightly coupled core components: they are less adaptable and demand more effort to maintain than more loosely coupled components (MacCormack et al., 2007, p. 26), and require a higher level of communication in the team developing them to achieve a given level of quality (Cataldo et al., 2006).

The direct connections between components can easily be recognized from a DSM, once the initial data gathering has been performed. However, direct connections are seen as insufficient to portray the effect of changes spreading further along the chain of dependencies, i.e., if one component is changed, some or all components that depend on it could also have to be changed.

The risk of changes in one component propagating to other components is called a component's visibility. The visibility can be examined by calculating the DSM's visibility matrix. One way of achieving this that has been used in literature is raising the DSM matrix to successive powers, with the nth power showing interactions through N steps; these are also called indirect dependencies. The power matrices are then summed for the final visibility matrix. The maximum power that should be used is the number of elements minus 1, as at the latest after that the paths become redundant. If a power matrix has only zeros in it, then the process can be stopped, as all following power matrices will also have only zeros. (Sharman and Yassine, 2004; Warfield, 1973)

(MacCormack et al., 2012, 2010, 2007, 2006) have extensively used the visibility matrix in examining the structure of software products, with component modularity being the focus in (MacCormack et al., 2010, 2007). They define the concepts of *fan-in* and *fan-out visibility*: fan-in visibility (FIV) means the "dependencies that flow into the

component", and fan-out visibility (FOV) conversely "those that flow out from it". By calculating both types of dependencies from the final visibility matrix, they divide components along the two axes into four categories: core (high FIV and FOV), shared (high FIV, low FOV), peripheral (low FIV and FOV), and control (low FIV, high FOV). "High" is defined as being at least 50% of the maximum value in the system. (Baldwin et al., 2013; Lagerström et al., 2013a, 2013b) use the same concept of visibility in their research, and find further evidence to support the four-class taxonomy. (Baldwin et al., 2013) additionally propose an algorithmic approach to recognize the four classes from the visibility DSM.

It is easy to see the possible managerial implications of component visibility. A high FOV means a high risk of change propagation, which could mean unexpected and costly additional work. Especially the core components are problematic, as they also have a high FIV; thus, changes from several components could lead to a change in a core component, which in turn will cause the change to spread to many other components. Therefore, efforts to make the "core" smaller, though risky, can improve the modularity of the whole system.

Yet, here the difference between binary and numerical DSMs will have to be considered. If a visibility matrix is binary, the each mark means a 100% chance of change propagation, which, intuitively, is overly pessimistic. Therefore, e.g. (Sharman and Yassine, 2007, 2004) suggest using probabilities of 0% to 100% in the DSM; in this case, the matrix multiplication takes into account the diminishing of the probability the further along the change path a dependent component is.

However, giving accurate estimates for the change probabilities is hardly a trivial task, especially given the inclarity over suitable proxies, as explained in chapter 3.3. Therefore, the simplicity of just using binary connections could be preferable. Furthermore, even if the visibility matrix gives an exaggerated estimate of changes spreading, the impacts will have to be verified in any case, which means work. Thus, even a binary visibility matrix will give insight into the amount work a change will require.

(Sosa et al., 2007b) propose an alternative way to calculate component modularity, based on direct and indirect dependencies and come components being on more change paths than others. They also take into account interaction weights, and thus present a wider view to component modularity. Finally, some of the problems with the probabilites can also be avoided, if the components are separated from component interfaces. If a program has an interface for other programs to use, a change in the internal structure of the program will usually not require changes in other programs. For this reason, it can be beneficial to present the program and the interface as separate elements in the DSM (Avritzer et al., 2010).

3.4.3 System modularity

Defined very coarsely, system modularity stems from organizing the system components into logical groups. The modularity of systems has been the subject of a large body of research, yet with little unanimity on key concepts such as the definition, meas-

urement or value of modularity (Gershenson et al., 2004, 2003; Schilling, 2000). Thus, these will be discussed next.

On the subject of definition, this thesis follows the two ideas of modularity presented by Baldwin & Clark (2000, pp. 63–64), which are quite widely accepted (MacCormack et al., 2012). Firstly, components should be interdependent within modules and independent across modules. This is similar to the loose and tight coupling discussed with regard to singular components in chapter 3.4.2.

Secondly, "the complexity" of an element "can be isolated by defining a separate abstraction that has a simple interface" (Baldwin and Clark, 2000, pp. 63–64). Thus, if a module is identified, it can be hidden behind an interface to limit the effects of changes. This, along with the first idea, allows changes to be made to a module without affecting other modules, provided the interface remains the same.

With regard to value, (Baldwin and Clark, 2000) argue that the benefit in modularity is that it allows independent development and substitution of modules, thus creating real options for the development of the system. Following general option theory, they propose that a portfolio of options is more valuable than an option on a portfolio. Thus, the ability to accept or reject separate modules makes the system consisting of the modules more valuable than if the decision would have to be made on entire systems. On the other hand, modularity also carries costs; in addition to the self-evident costs of designing the modules, (Schilling, 2000, p. 316) for example presents the idea of "synergistic specificity", which means the performance gains achieved by higher integration of components; this is lost with increased modularity.

Therefore, the usefulness of modularity depends on the context. In the setting of information systems, increasing modularity through information hiding has been very popular since (Parnas, 1972) proposed it as a tool in his seminal paper. Afterwards it has become a central notion of the object-oriented programming paradigm. By limiting the information visible from one module to another, concurrent work on separate modules is possible. For instance, (MacCormack et al., 2006) present the case of the Mozilla Firefox browser, that underwent an overhaul in order to make the software's architecture more modular. This was done in order to facilitate open-source development, and was arguably very successful: the new architecture made possible to concentrate on smaller pieces of the software, with development work afterwards taking place on many fronts simultaneously.

Information hiding through interfaces is also arguably very easy in software, as components and interfaces consist solely of information. Of course, industry-wide standards also exist and are slower to change, but in the context of a single product or company, major revisions of the product architecture should be simpler than in the physical realm.

Similarly, in the frame specified for this study, EA and AA, it can be argued that the benefits gained by increased system modularity far outweigh the losses in efficiency. As discussed in chapter 2, the aims of EA are for example component reuse and interopera-

bility between business units. These are only attainable, if there are well-defined standards and interfaces that can be used for connecting information systems together.

System modularity in the DSM context is very much tied to clustering, as discussed in chapter 3.2.2. The first idea of modularity (Baldwin and Clark, 2000, p. 63) presented above is the primary aim, with two separate effects in a DSM. Firstly, one seeks to form groups of components with few marks outside module boundaries, with the marks inside the module boundaries acting as the base for grouping the components together. In the case of a numerical DSM, some weighting scheme will have to be considered, and the aim becomes minimizing the strength of the connections outside module boundaries. Secondly the size of the clusters should be minimized. These two aims are conflicting, so a trade-off will have to be considered. (Browning, 2001, p. 294; Eppinger and Browning, 2012, p. 25)

The second idea of modularity (Baldwin and Clark, 2000, p. 64) is a tool for achieving loose connectivity in the system. By creating interfaces, 'design rules' as discussed earlier, modules can be disconnected from each other. Elements that do not fit into other clusters due to wide-reaching connections can be organized into integrative (Sosa et al., 2003) or bus (Sharman and Yassine, 2004) clusters, that interact with many other clusters; these can be thought of as interfaces.

It may also be useful to allow elements to belong to more than one cluster; Sharman & Yassine (2004, pp. 43–44) call this pinning, as the relations to more than one cluster 'pin' components in place between clusters. In this case, the shared elements represent a natural interface between the sharing modules.

However, it must be noted that simply clustering a DSM does not always guarantee a sensible module division, especially if clustering is done by an automated algorithm; other considerations than the marks in the DSM will usually have to be considered (Gershenson et al., 2004; Newcomb et al., 1996; Schilling, 2000). In short, the module division should be based on cohesion in addition to coupling: the elements in a module should have more in common other than a lack of connections to other modules (Dreyfus, 2009, p. 38). Popular clustering algorithms are often only capable of considering the coupling between modules, and as such, human input is often required to validate the usefulness of the results. However, clustering can give ideas for organizing the product architecture.

The next question is, how to compare two architectures on modularity, ergo, how to measure it. However, this is subject to the same discussion than the definition of modularity. In the scope of DSM, authors have approached this question in a variety of ways. Firstly, there is the real options theory proposed by (Baldwin and Clark, 2000) and used by e.g. (Lopes and Bajracharya, 2005; Sharman and Yassine, 2007; Sullivan et al., 2001). The real options value is interesting, at it presents a single value for the whole architecture, which would make for easier comparisons of prospective architectures. In addition, (Sharman and Yassine, 2007) argue that many other metrics present a very naïve way of analyzing architectures. The metrics look only at interactions between components and a specific time point, and as such fail to address for example the im-

proving systems value, which is one of the reasons why modularity is said to emerge in the first place.

However, there are several hindrances to using this approach. The real options theory necessitates several assumptions when calculating the value of the modules, such as a normal distribution of expected values. Additionally, an estimate of the "technical potential" would have be proposed for each of the modules. This would have to rely on human expertise, and therefore introduces significant uncertainty into the calculations. Authors using this approach don't always use very convincing estimates either, relying on ex-post information (Sharman and Yassine, 2007, p. 164) or just saying that in a real-life situation, "a designer would generally have to justify the choices of parameter values more convincingly" (Sullivan et al., 2001, p. 106).

Secondly, the measures of component modularity can be aggregated into system-level measures. (MacCormack et al., 2012) define the dependency density the final visibility matrix as "propagation cost". This means the average number of components affected, when changing a single component. (Sharman and Yassine, 2004) use scatter plots of component visibility compared with the number of dependencies, with more modular architectures exhibiting certain patterns.

Thirdly, different measures have been specified for with system-level analysis in mind, in e.g. (MacCormack et al., 2006; Sosa et al., 2007a). An aggregated value is of use especially when using automated clustering algorithms, as these need an objective function to compare architectures. However, (Hölttä-Otto et al., 2012) find that most existing measures of modularity are inadequate, as they do not take into account the existence of bus components. According to the authors, this is a major shortcoming, as buses are common in real-life systems, a claim verified also by other writers (Sharman and Yassine, 2004; Sosa et al., 2003). They recommend a measure developed by (Yu et al., 2007), which can distinguish a bus module. On the other hand, they recognize that a custom metric reflecting exactly the desired business benefits from modularity can be the best alternative (Hölttä-Otto et al., 2012).

Another alternative is that the bus or buses are recognized beforehand, and calculations are performed without these bus components. (Sosa et al., 2003) present one idea for identifying the bus components, by using a chi-square comparison in order to differentiate between modular and integrative subsystems, in other words, modules and buses. However, their approach can be time consuming, as each component would have to be tested against each other component. (MacCormack et al., 2006) set a "bus threshold", which means a percentage of other components having connections with a certain components. Finding an appropriate threshold value is claimed as depending on context, however.

All in all, there is no consensus of the "best" measure of modularity, even in the DSM setting. Thus, the problem should be approached by testing alternative measures and visualization methods, and seeing which has the best "predictive value" (Sosa et al., 2007a). In addition, there are other concerns, such as ease of use and the cost of implementation.

4 RESEARCH APPROACH

This chapter lays out the specific approaches to be used in later chapters. In addition, basic information on the context of the study is given in the form of an overview of the case company, ABB Motors & Generators, and the collection of data for the study.

4.1 Research setting

4.1.1 The case company: ABB Motors & Generators

The research for this thesis was carried out at the Pitäjänmäki, Helsinki site of ABB Motors & Generators business unit. It is a part of ABB Ltd., a global power and automation company founded in 1988 and headquartered in Zürich (ABB, 2012). As the name implies, the ABB Motors & Generators products include generators and electric motors in various sizes. Separate financial details are not reported for the business unit, but it is a large organization with operating sites at several countries on nearly all continents.

To simplify notation, ABB Motors & Generators will henceforth be referred to as simply ABB. If there is need to refer to the whole group, it will be called ABB Ltd. Any other business units will be referred to with their entire name.

The specific context of this study is the application portfolio of the order-delivery process at the Pitäjänmäki factory, which is being developed as part of ABB's AIMO project. This context is relevant for defining the objects of interest for the analysis; this is discussed in chapter 4.1.2. Other details concerning ABB can largely be dismissed as unimportant. The applications in use have of course been chosen and developed with regard to the particular type of operation at ABB, but the main interest of this thesis is the inner structure and management of the application portfolio, not specific applications. Therefore, questions such as how well the applications function in their intended role are outside the scope of the study.

4.1.2 Current phase of EA analysis at ABB

At ABB, a modified TOGAF (The Open Group, 2011) has been the framework of choice for directing the EA work. At the moment and in the words of the TOGAF ADM model, work has proceeded to the level of information systems architecture. In other words, business goals and processes are taken as given in this thesis; they have been considered previously to the extent that is considered useful.

As said before, ABB is interested in utilizing the DSM tool in the analysis. This is seen as a promising approach, as many EA frameworks already use square matrices similar to DSM as modeling objects. A square matrix is a part of the U.S. Department of Defense Architecture Framework, or DoDAF in short, as artifact SV-3, "Systems-Systems Matrix" (DoD, 2010). The artifact contains information on interactions between software systems in use. However, the section covering its use is very general, offering only ideas on what can be modeled in the matrix; advice for practical use is not presented. The TOGAF framework (The Open Group, 2011) gives some more detail as part of the "Application Interaction Matrix". The aim is to "depict communications relationships between applications". In essence, the matrix is a visual presentation of previously collected catalogues of applications and interfaces between them.

Figure 4-1 shows the TOGAF ADM and elaborates briefly the two areas of interest of this thesis. The DSM is shown at the bottom, and the sections to be analyzed with the

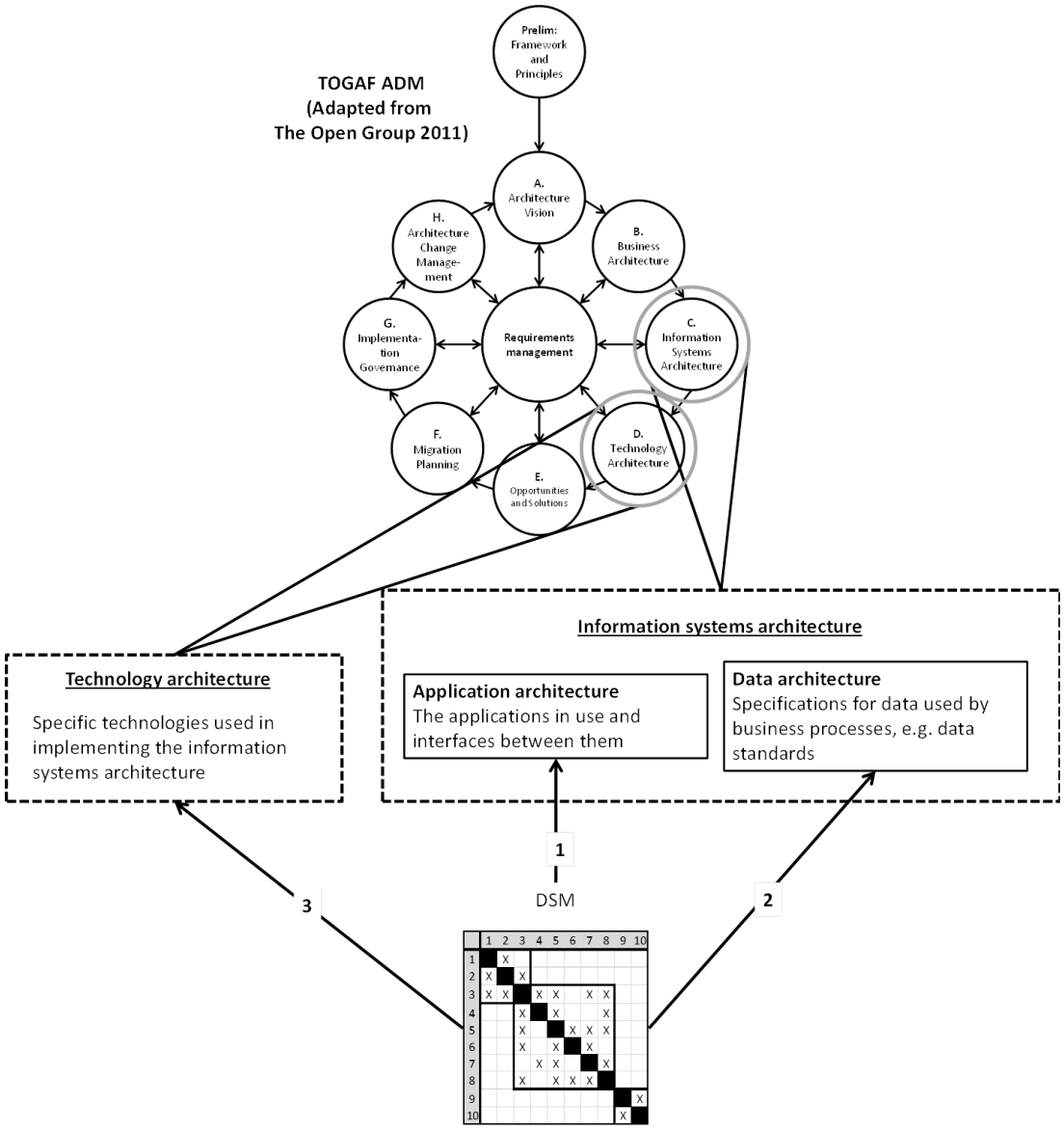


Figure 4-1: Areas to be analyzed with the DSM

DSM are marked with numbered arrows. It is immediately obvious that focusing on just these areas of the model leaves out important subjects, such as the business architecture and implementation of the changes. However, this focus represents ABB's interest. Firstly, with regard to available resources, the business architecture was decided to be ruled out of the analysis; involving it would have meant too much work. Secondly, as discussed in chapter one, the aim of this study is to generate theoretical proposals for architecture development, and therefore concerns of practical implementation are not of interest at this point.

A specific concern to address overall is flexibility, defined as the ability to adapt the AA to "changing business requirements [--] quickly and without major efforts" (Schmidt and Buxmann, 2010, p. 173). The value of IT flexibility has been discussed also in literature, with findings that flexibility for example increases the strategic alignment of IT (Ness, 2005, p. 12) and contributes to competitive advantage, albeit with increased costs (Byrd and Turner, 2000, p. 195). In ABB's case this tradeoff between IS costs and flexibility is well-known, but the main interest is in the flexibility.

(Duncan, 1995) further divides flexibility into three concepts: the ability to create connections between (connectivity) and share information across (compatibility) IT components, and to add, modify, and remove IT components with no major, large-scale effects (modularity). These aims are represented differently in the architecture layers depicted in picture 2-2. This is discussed below. More information on these questions and approaches on using DSM in the analysis are discussed in chapter 3.

First is the application architecture (AA). To be more precise, the focus is on the concrete portfolio of applications currently used to produce information to business processes; another concept in literature with largely the same contents is the enterprise information system (EIS) (cf. Schmidt and Buxmann, 2010, p. 169). Previously, at ABB there have been problems with changes made to one program spreading to other programs in an unanticipated fashion, causing extra work and costs. Thus, there is need to study the connections between programs in use, and produce information and tools to make the application portfolio more manageable. In the terms of (Duncan, 1995), the interest is in the modularity of the AA.

Second, and closely related, is the analysis of the data architecture (DA). This architecture domain specifies, for example, data standards that applications must adhere to in order to ensure interoperability. These could be implemented as globally defined interfaces, with the aim of for allowing different sites room for optimization, while still giving global guidelines.

One idea that ABB is interested in investigating is dividing the applications into "business modules": groups of applications that are used in the same phase of the order-delivery process. This is elaborated in Figure 4-2 below. For example's sake, the order-delivery process has been divided into five phases.

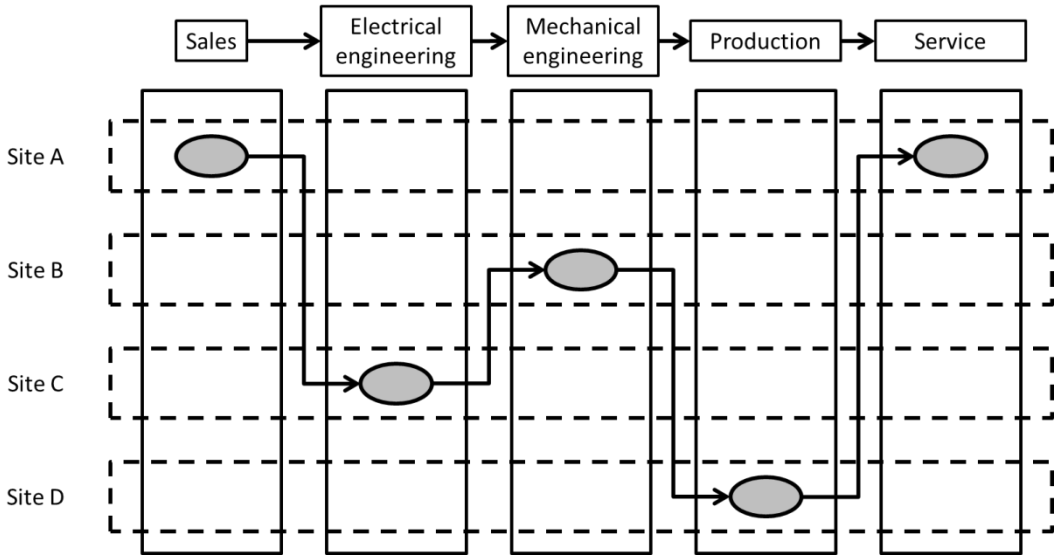


Figure 4-2: Business module concept

The ideal situation would be that each phase of the process could be carried out at a different site, if so desired; this is represented by the grey ovals in the picture. This would make the process very flexible, in the sense that the workload could be divided between sites based on current resource utilization or expertise, for example.

From an information systems point of view, the problem is that each site could have rather different applications in use for each phase of the process, with different data standards. As such, moving information from one site and process phase to another is not entirely simple.

This is a very common situation when a company grows through mergers and acquisitions, as each previously separate company could have a lengthy information systems history of their own. However, this should not be considered solely a problem either, as the information systems at different sites will most likely have been developed to respond to customer demands of that precise site. Simply forcing a site to implement the same applications as all other sites could mean that this sensitivity to local customer demands is lost, not to mention the inevitable resistance to change from users. Additionally, an important objective of EA is to allow local management of information systems, while providing enterprise-wide standards for interoperability.

Thus, the idea is not to define the exact contents of the business modules, but rather the interfaces between them; this way moving information between sites is made possible. Similar initiatives have been tried in other companies as well: (Schmidt and Buxmann, 2010, p. 174), for example, report that many organizations in their study divided their application portfolio to decoupled domains, in order to increase EIS flexibility. With regard to (Duncan, 1995), this is close to the aims of connectivity and compatibility, as the strong interfaces will facilitate communication between existing sites and adding new sites to the network. This in turn adds to the modularity of the AA, as the applications can be switched and modified as wanted, as long as they conform to the interface standards.

With regard to the DSM, the approach is elaborated further below through examples. First, consider the following system architecture presented both as a graph and a DSM. The nodes in the graph and elements in the DSM represent specific software applications. The edges in the graph and marks in the DSM represent information flowing from one program to the other.

Figure 4-2 could represent an enterprise information system that has grown "organically". The system architecture could have been different in the beginning, but over time additional connections have been created between applications in order to increase system efficiency. However, at the same time, the system has become less manageable. First of all, making sense of the architecture is not an easy task, as the programs do not appear to form any logical groups. Replicating this architecture at a different site or connecting other sites to this architecture would also cause problems; some of the programs could be site-specific, and if different programs belong to different phases of a process, it would not be clear how to move information between sites at different phases of the process. Deeper analysis would also uncover a feedback loop (P9 → P5 → P2 → P1) that spans the entire system, which could cause unanticipated consequences if changes were made to a part of the system.

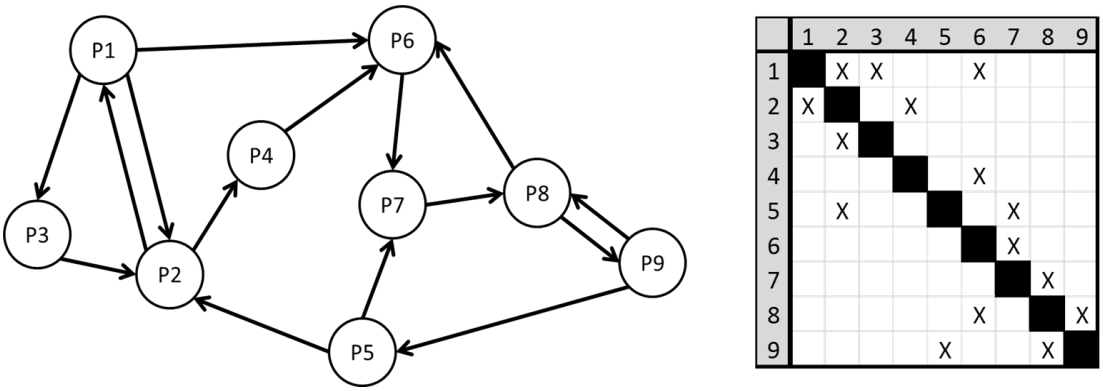
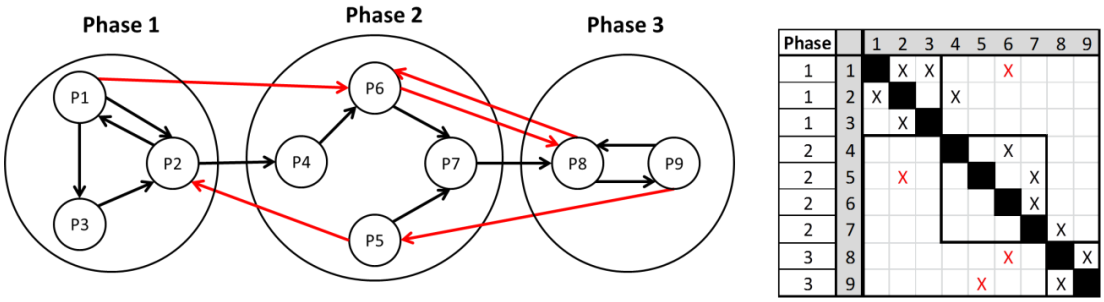


Figure 4-3: Initial architecture

The first step would be to add information on which business module each element belongs to, i.e., which part of the process being examined. For example's sake, let's assume that the business modules would be as presented in Figure 4-4 below.

Red color represents unwanted connections between programs. For example, if one site would like to replace all programs linked to one phase with a single program that has the same features, the optimal state would be that there is only a single connection between two business modules, as this situation would be the easiest to manage. Each additional connection that crosses module boundaries will cause additional difficulties when making changes. Changing the system to this direction could be accomplished by choosing or creating the desired connection between modules and eliminating the rest; the specific implementation will depend on system-specific qualities. However, feedback connections between modules, those below the diagonal in the DSM and outside module boundaries, can be considered unwanted in any case. If the system is divided



Phase		1	2	3	4	5	6	7	8	9
1	1		X	X			X			
1	2	X			X					
1	3		X							
2	4						X			
2	5		X					X		
2	6							X		
2	7								X	
3	8						X			X
3	9					X			X	

Figure 4-4: Programs divided into process phases

into specific process phases, feedback can be allowed within these process phases, but not between them.

Figure 4-4 represents one way of organizing the process information flow, with information flowing directly from each process phase to the next. An alternative way to set up the system would be to use an enterprise bus, similar to EAI, as discussed previously. This is pictured below in Figure 4-5. The bus structure could be interpreted as a global interface that each business module uses to communicate with others; for example, (Lamantia et al., 2007) studies two database programs that both adhere to the Java 2 Enterprise Edition (J2EE) specification, and uses the specification as a design rule to reorganize the architecture. Another example is the "enterprise service bus" presented by (Papazoglou and Heuvel, 2007) as a means to decouple software domains from each other. The bus could also be implemented as a data warehouse, which provides consolidated services to each business module.

In the first phase, ideas for DA development will be based on the AA: the data standards are looked for in the applications in use. For example, if an application interacts with two consecutive business modules, then it forms a natural interface between the two modules. On the other hand, bus-like or integrative elements are common in real-life architectures as discussed before, and these could form potential candidates for creating an enterprise bus. Basing DA on AA is by no means a unique approach: TOGAF groups DA and AA together, because they are of the mind that either one can be developed first. However, for a large and established organization such as ABB, this approach could be easier to implement. A revision of the data architecture would probably mean large changes to existing applications; on the other hand, putting the application architecture first means that, in the best case, no changes are needed, if applicable interface programs can be identified.

To be realistic, both approaches to building the interfaces will in all likelihood require changes to existing applications and connections between them; for example, looking at Figure 4-3, it is highly unlikely that all necessary information could be transported through the process if the connections marked in red were just severed. Thus, creating an interface between two consecutive business modules would probably require some kind of consolidation of information before giving it over to the next business module. Likewise, even if an enterprise bus could be based on existing elements, the interfaces to and from the bus will probably have to be modified to allow transporting

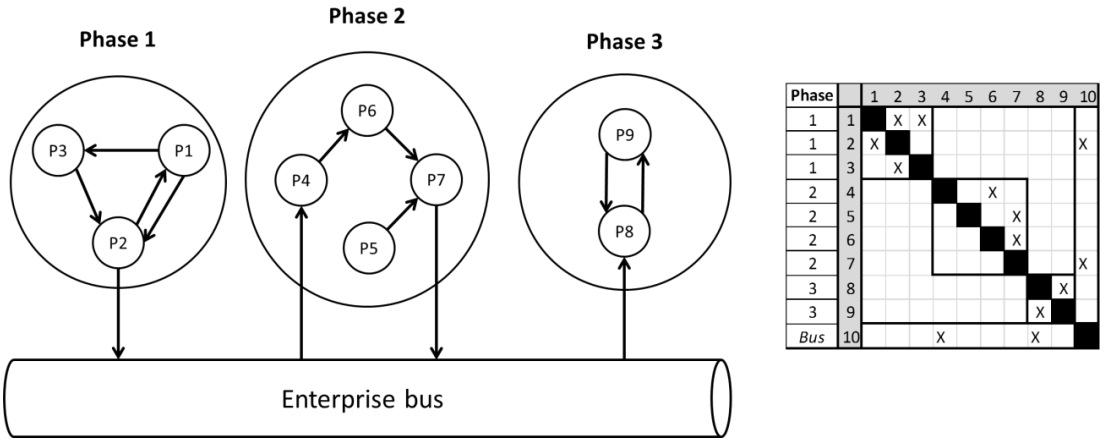


Figure 4-5: Connections through enterprise bus

all necessary information. However, looking back at chapter 1, the objective of this study is a tool for generating suggestions; this is followed by manual editing to ensure practical feasibility.

Thirdly, the analysis will also need to consider the technology architecture (TA), i.e., the specific technologies used to implement the information systems architecture. The main interest here is the nature of interactions between the applications. The basis for defining interactions at ABB has been logical; there is an interaction between two applications, if one application supplies information to the other application. This was chosen as the approach in order to make the concept of the interconnections more understandable during data collection. At the moment, an interconnection may have been implemented in any of a number of different technologies. They may even span several steps, yet be portrayed as just a single interconnection. While this represents well the logical view to the interconnections, the differences in the technological implementation could impact the strength and complexity of the connection, and thus should be portrayed.

4.2 Research methodology

The research will be conducted as a single case study. Although single case studies generally have trouble uncovering generalizable results, they are appropriate for studying new topics (Eisenhardt, 1989). As discussed in chapter 3, there are few works discussing using DSM in EA context. In addition, these papers present ideas for analysis of EA, they often utilize only one approach and leave open the question of how to specifically use the approaches in developing EA and AA (cf. Lagerström et al., 2013a, 2013b). This thesis will try to present ideas for this question, although further research is surely required in order to verify the usefulness and the applicability of the findings outside the specific context studied here. Another, a more practical reason for using case study methodology is the easier access to the studied organization, due to the close connection to the context.

Single cases as opposed to multiple-case studies are usually not preferable, as the challenge of generalizability is even more pointed (Yin, 2003). However, even single case studies have important uses as motivation, inspiration and illustration of theoretical claims, if the conceptual side of the research has merit on its own. (Siggelkow, 2007) Indeed, the product of this study will be mostly conceptual: a framework for providing information on the areas of interest discussed in the previous chapter. This framework will also be used in building a practical tool or step-by-step guide for performing the analysis.

The framework will be constructed by combining information gathered from two sources. Firstly, the literature presents information on the two foci of the study, EA and DSM. Due to the methods of analysis being strongly preordained, the literature will have a strong impact on the end results of the study. Secondly, the information needs of ABB will also have an eminent role. As discussed in chapter 3, the DSM can be used in quite a diverse array of ways, with no approach clearly better than another one. Therefore, the fitness for the purposes of ABB will be a guiding factor in deciding how to implement the tool.

Thus, this thesis follows a partly deductive, partly inductive approach. The deductive side comes from the strong role of the existing literature. The inductive side, on the other hand, is due to the close tie to the research context. (Saunders et al., 2009, p. 127) The inductive side follows from the ties to ABB's needs. A solely deductive approach runs the risk of being out of touch with the highly contextual managerial needs of an organization. Therefore, the inductive element complements the theoretical basis.

The case study consists of subsequently testing and developing further the framework using data from ABB. The nature of the study is largely exploratory. The main interest is in developing the framework so that it matches the needs of ABB, and could be used in other settings as well. Further, this thesis will not consider the testing of whether the framework leads to increased AA flexibility. While this is highly important with regard to ABB's needs, testing the approach would require comparing the architectures over several evolutions of the AA. As even one round of changes could take months to implement, this longitudinal focus is not possible within the bounds of this thesis.

4.3 Proposed conceptual framework

The proposed concept is presented in Figure 4-6, based on the literature review and ABB's areas of interest presented in chapter 4.1. The main products of the approach are at the bottom of the figure. These are the theoretical development suggestions for the architecture, based on communications from module to module or using a bus structure. The other parts represent inputs needed to form the development suggestions. Direct inputs are presented with a solid border, and indirect inputs, the inputs of inputs, if you will, are presented with a dashed border. These will be discussed next.

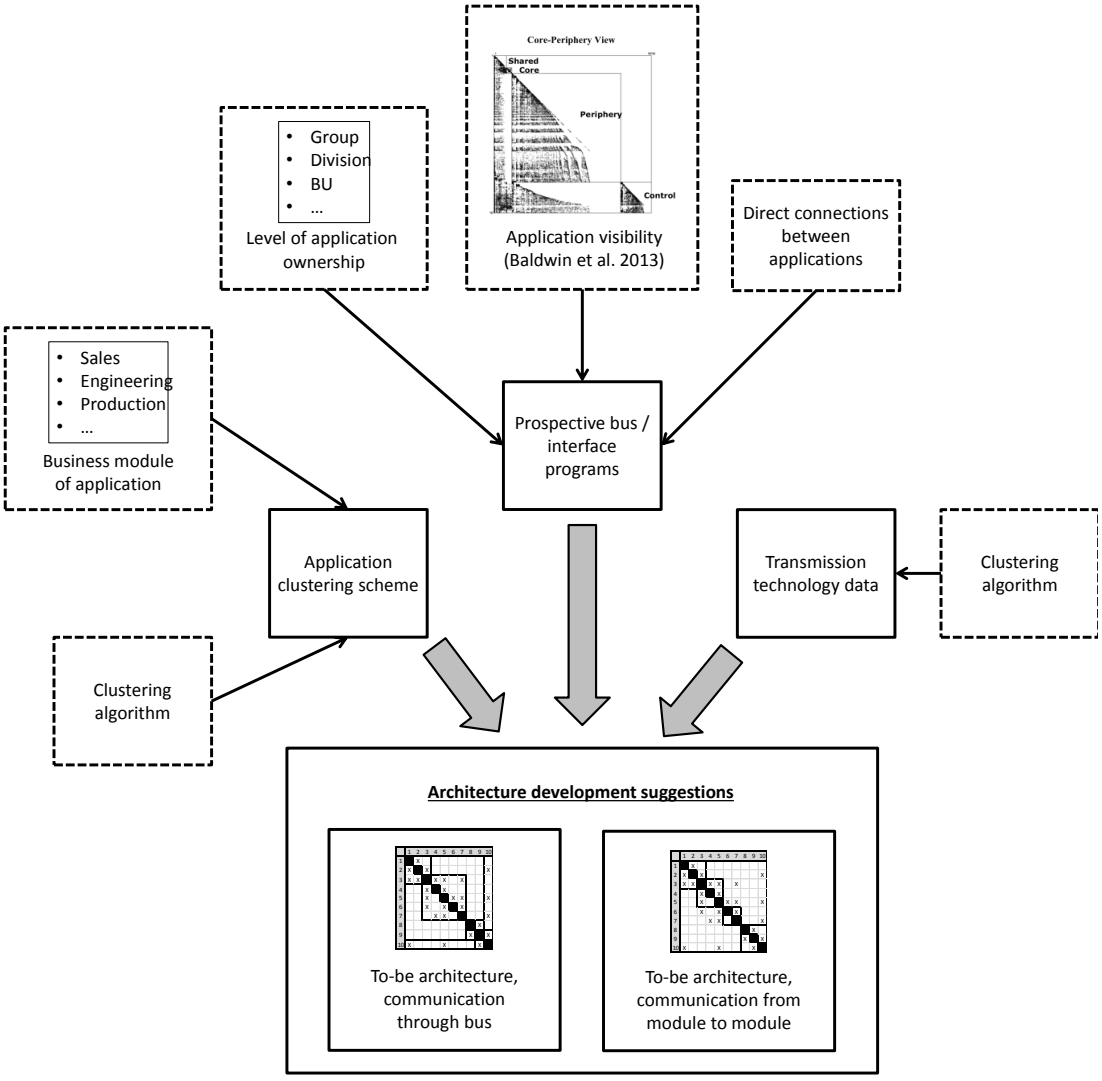


Figure 4-6: Proposed approach

Firstly, to create the development proposals, a decision will have to be made on what application to use as interfaces between the modules. In addition, to differentiate between the bus-based and the module-to-module architecture, the bus elements to be used are needed as input. As discussed in chapter 3.4.3, many real-life systems possess a bus-like structure, in that there are modular components that can be grouped into independent clusters, and then integrative components that logically form a backbone for the architecture.

Three approaches can be used here. Firstly, one should naturally inspect the marks in the DSM. Promising bus elements are those that have many connections all along the DSM, horizontally or vertically. These represent problems for any kind of clustering scheme, as discussed previously. As for real-life applications, setting some elements as buses can make for much more viable architecture plans than if all elements would be set to separate clusters. For example, looking at Figure 4-4, the marks drawn in red all represent unwanted interconnections. Getting rid of any of these could necessitate a many-month project of its own, due to the technological complexity of the connections.

Therefore, setting some elements as buses enables focusing the architecture development on a smaller portion of the EIS.

Secondly, the level of ownership of the application sets limits on what kind of and how extensive modifications can be performed on an application. For example, a group-wide ERP system will be much more rigid than an application used only at one business unit, as there are higher-level policies controlling the use of the application. More widely adopted applications also make for good interfaces, as they are to a large extent the same at many sites.

Thirdly, the visibility matrix discussed in chapter 3.4.2 can be used to estimate risks of making changes to applications. For example, applications classified as "core" depend on and are depended on many other applications; thus, making changes to these applications will require more work in checking and modifying connections than if the changes are made to "peripheral" applications. In the words of (Baldwin et al., 2013), applications with a high FIV make for logical interface applications: changes to these applications carry a high risk and call for much work, and therefore it is easier to make other, more easily modified applications comply to these.

The second question is what kind of clustering scheme to use. In other words, on what grounds should the applications be grouped? As discussed, system modularity consists of two viewpoints: cohesion and coupling. Both will be brought into the model. The former, cohesion, comes in the form of source data on the business modules. This will create groups of applications that are logically coherent. The latter, coupling, will be included as the results of an automatic clustering algorithm. This corresponds with the notion of modularity as presented by (Duncan, 1995). The clustering schemes proposed by an automatic clustering algorithm will be based strictly on the connections, with the aim of ensuring that as few connections are left outside the modules as possible.

A question related to the automatic clustering is what method to use. There are a number of different combinations of modularity metrics and functions that attempt to optimize the value, so a choice will have to be made for the practical implementation. These are discussed in more depth in chapter 4.4.3.

However, many metrics of system modularity fail to take into account the existence of the bus, which often leads to an incorrect system modularity valuation of the architecture, and can result in non-optimal solutions when using automated clustering algorithms. Therefore, one proposed workaround is to ignore the bus elements during the metric calculation. (Hölttä-Otto et al., 2012, p. 804) Identifying the bus elements is thus a logical starting point, if a using a bus is desired.

The MDL metric presented in (Yu et al., 2007) does account for the bus, which remedies the problem somewhat. However, firstly at this point it is not desirable to be limited to just one metric, as considerations such as ease of use could render the metric unusable at a later point. Secondly the MDL metric accounts only for direct dependencies, which according to e.g. (Baldwin et al., 2013) is not sufficient; the indirect dependencies have to be taken into account as well. Thirdly, identifying the integrative

bus elements serves another purpose as well, namely proposing suggestions for the interface applications to link the other applications. This ties into the topic of design rules and EAI discussed in chapter 3.4.1. The interface applications can then be used as basis for the development of the DA.

Thirdly, the transmission technologies should be brought into the DSM, if they are to be analyzed. For example, if different transmission technologies are thought to have different complexities and risks of change propagation, then the technologies should be represented, at least for forming the practical development plan.

At this point, these inputs are thought to be useful in forming the architecture development suggestions. The latter part of this thesis will test this conceptual model to refine, what data will be included in the final theoretical suggestions, with the aim of offering ABB a concrete tool for AA analysis. The questions that remain to be answered will be discussed next.

4.4 Issues to be considered

4.4.1 Representing the inputs in the development proposals

In Figure 4-6, many inputs are presented that are hypothesized as being useful in creating the architecture development suggestions. However, at such an early stage of development, it is not clear how to incorporate them into the final suggestions, and whether they bring anything useful to the suggestions at all. These will have to be examined, taking into specific account the views of the future users of the proposals.

A related subject is how to use the clustering schemes created by both using the business modules and through a clustering algorithm. These form a central part of the development suggestions, as they are the starting point based on which the interface suggestions are created. A specific area of interest is how to combine the information provided by both methods of creating clustering schemes.

4.4.2 Comparing as-is and to-be architectures

In order to value different plans for to-be architectures and compare as-is architectures at different times, a way measure the architectural suggestions will have to be developed. Different measurement schemes were discussed as part of the literature review; however, with regard to the development goals discussed in chapter 4.3, it is not entirely clear if these would be useful and how they would need to be modified if implemented.

Some types of metrics have been identified as initially interesting. For the core and peripheral components, the sizes of the different component classes (core, shared, control, peripheral) are of interest. Especially the size of the core is pointed out as important, as it has a great effect on the risk of change propagation (Baldwin et al., 2013; MacCormack et al., 2010). Thus, a to-be architecture with a smaller core is seen as an improvement over a current as-is architecture.

For system modularity, there are many potential measures, presented in e.g. (Hölttä-Otto et al., 2012; MacCormack et al., 2006; Sharman and Yassine, 2007; Sosa et al., 2007a). Due to the difficulties of using the real options theory, The MDL measure (Yu et al., 2007) seems to be the most promising way forward. However, as discussed it might not be implementable due to reasons of practicality. Therefore, at this point, the MDL is set as the initial choice, but the inspection might have to be widened later on.

If additional, ABB-specific views into the architecture are needed, these are very likely to require custom measures that capture the point of interest. All in all, a small, representative selection of measures is probably the solution most likely to be implementable as the metric for architecture flexibility. However, for managerial purposes, the best-case scenario would be a single measure taking into account all the different presented views to the architecture. If this is to be implemented, either one measure will have to be chosen above the others, or a way of aggregating the measurements will have to be developed.

4.4.3 Implementing automatic clustering

A small DSM, of typically less than 50 elements, can usually be clustered manually in e.g. spreadsheet software, as the complexity is still understandable. For larger matrices, automatic clustering algorithms are more desirable, and several options for this have been developed. (Börjesson and Hölttä-Otto, 2012) Of course, the best option would be to analyze each variation of the architecture and choose the best one. However, as the size of the DSM grows, the computational requirements quickly become unbearable. For this reason, many heuristic approaches in the form of clustering algorithms have been developed. (Yu et al., 2007, p. 99)

A clustering algorithm is composed of two separate pieces: the objective function calculated in order to compare architectures; and the method to arrange the DSM elements to different architectures. Authors usually present their approach using a specified combination of the two, but theoretically any combination of metrics and algorithms presented in literature could be implemented.

The algorithms considered here are based on those presented in (Eppinger and Browning, 2012). Desired qualities of the algorithms are firstly the quality of the results they are capable of producing, secondly the runtime. The time is of course affected on other factors than just the algorithm, such as the hardware in use and the technology used to implement the algorithm.

The first algorithm is the IGTA, presented in (Thebeau, 2001), building on the work of (Idicula, 1995) and (Fernandez, 1998); hence the acronym (*Idicula - Gutierrez* Fernandez - *Thebeau Algorithm).* It has been widely used in literature, both as a tool (MacCormack et al., 2006; Pimmler and Eppinger, 1994) and an object of study (Sharman and Yassine, 2007, 2004). The algorithm searches for the optimal architecture by choosing an element randomly and having the existing clusters bid for the element to join the cluster. The bid is calculated as the marginal reduction of the objective function attained by moving the chosen element to a cluster. The search algorithm then introduc-

es two random variables to keep the algorithm from getting stuck on a local optimum value, when it is not the global optimum.

(Thebeau, 2001, p. 50) cites the run time as 10-30 minutes. With the computer hardware development between 2001 and 2013, the time taken can be expected have lowered significantly. However, in more recent literature, (Börjesson and Hölttä-Otto, 2012) argue that due to the heuristic nature of the algorithm, it will have to be run a large number of times to be sure of the result. They cite the time taken to run the original IGTA algorithm 10000 times on a DSM of 57 elements as about seven hours. This would be a considerable investment from the user, and "does not invite 'tinkering'" (Börjesson, 2012, p. 97) with the DSM. For this reason they present a modified algorithm that is significantly faster, up to by a factor of eight, and produces better results, as measured by their objective function. In order to achieve these results they have to sacrifice allowing an element to belong to many clusters simultaneously, but they propose also other augmentations that can be used without this loss.

(Yu et al., 2007) propose using a genetic algorithm for the search. A genetic algorithm uses mechanisms familiar from biological evolution, aiming to simulate natural selection, or the proliferation of the most suitable solutions. The alternative solutions are depicted as groups of genes, or chromosomes, which are then combined in random pairs. Crossover and mutation of genes in the descendant chromosomes are introduced to bring variability to the gene pool. This cycle simulating mating is then repeated a desired number of times; usually a higher number produces better and more consistent results, but on the other hand consumes more time (Goldberg and Holland, 1988). The algorithm used by (Yu et al., 2007) is cited as taking "about a day" to run for a DSM of 60 elements, although again the hardware has evolved between then and now.

(Zakarian, 2008) presents an algorithm for clustering non-binary DSMs. The algorithm itself is quite simple and very similar to IGTA, in that the elements 'bid' to be included in a module. The bids are calculated solely from the weight of interconnections outside modules, and only the modules that have connections to an element bid it. Subsequent iterations typically result in a smaller number of larger modules; the 'optimum' architecture from the objective function's point of view is a structure consisting of just one module, as then there will be no interconnections outside modules. Therefore it is easy to see that too many iterations of the algorithm will defeat the purpose of clustering instead of producing better results, as in many other algorithms. On the other hand, each iteration will represent a different view to the architecture, and thus may contain interesting information. In addition, the author claims the runtime of one iteration to be only a few seconds.

All these represent viable alternatives to implement automatic clustering in the tool. Thus, they will have to be tested in order to assess, which is the most suitable. This will be considered in the case study.

4.4.4 Depicting the interactions

The nature of the interactions depicted in the DSM will have to be decided. As discussed, the specific technological implementation of the interconnections could have an impact, and might therefore need to be taken into account. Also, the literature did not present any definitive answers on what type of interconnections to use in the DSM would be the most informative.

From the literature, the approaches presented by (Fenton and Melton, 1990; Offutt et al., 1993; Xia, 2000) seem like the most promising, as they give ideas for making different interconnections comparable. However, they were meant for analyzing singular software systems, and so it is not clear if they can be adapted for AA analysis. This will be investigated.

Other options for depicting the interconnections include of course the simple existence of an interconnection, thus making the DSM binary. While this is not the optimal solution for reasons discussed previously, it will still give information and is the simplest solution to implement. Yet another option is to use a custom interconnection measure reflecting ABB's context.

5 INITIAL SITUATION OF THE STUDY

This chapter discusses the preliminary phases of the study. Firstly, information on the collection of data is presented. Secondly, the reasons behind the choice of technology for implementing the tool are considered. Lastly, the initial DSM created from the source data is presented and representing the interactions in the DSM is discussed.

5.1 Study practicalities

For the most part, data used in this thesis were collected as part of ABB's AIMO project between February and May 2013. The collection was performed using a special application built on Lotus Notes called MoGeApps. Employees responsible for applications filled two types of forms: application forms containing basic information on the applications themselves, and interface forms describing interconnections between applications. In ABB's case, an interface is defined as one application providing information to another. Dividing the information system to two separate catalogues follows the approach set by TOGAF, which proposes the two catalogues as the starting point for developing the information systems architecture.

The author was not part of this phase of data collection, and therefore the part of the research using these data will be document-based. During the duration of this study additional data were collected on business modules and levels of ownership of the applications; these will be discussed separately in chapter 5.5.1.

There are obvious risks with relying on this type of unsupervised human input for data collection, as all responders might not understand the instructions or the questions in a similar fashion. The situation is largely the same as for self-administered questionnaires.

Of course, there are ways of lessening the risks somewhat, by using e.g. choice lists instead of free text fields. In general the forms for collecting data on applications were constructed to include as few free-text input fields as possible. More importantly, the reliability of the data will not affect the results of this thesis very strongly, as the contribution is mostly conceptual. Unreliable data will weaken the usability of the proposed tool, but it is a concern separate from this thesis.

The practical part of the study was conducted between September and November 2013. As the study document-based, there was no real significance to where exactly the study took place, but for the most part, the work was done at the ABB Pitäjänmäki site. The source data underwent changes during the course of the study, such as applications and interfaces being added to and removed from the catalogues. However, the changes

will pose no threat to the validity of the findings, and in subsequent chapters the date of the source data used in the DSM and any other changes will be identified.

The practical steps of this study phase consisted of development of the tool in close cooperation with the supervisor of the study from ABB. The supervisor has extensive knowledge of the application architecture at ABB, and will also be the person responsible for the continued analysis of the architecture, at least initially. Thus his views could be presumed to represent well the true information needs for the architecture work. In addition, two meetings were held with the study supervisors from both TUT and ABB, and the business unit IS manager, who has been involved in this project from the beginning. The meetings acted as check-ups on the progress of the project and were meant to bring together all the views pertinent to the study.

The opinions that are presented on the usefulness of different approaches in the following chapters are based on the discussions pictured above. Thus they represent the views of only three people, which will of course set restrictions on the generalizability of the results. However, the input received on the tool can be assumed to match the needs of ABB quite well. In addition, as the study concerns more methods of analyzing the architecture than actual results, the findings might be possible to adapt to other organizations as well.

5.2 Choice of technology

One of the first choices to make was which technology to use top implement the tool. This will be discussed next. First, the requirements set on the tool will be discussed. Afterwards, three identified options for implementation technology are presented: dedicated software, Microsoft Excel and MATLAB.

First one should think of setting where the tool will be used. The employees using the tool will primarily be those tasked with EA responsibilities. It is presumable that they will not have much interest in acquainting themselves with specifics of the program code, and so attention will have to be directed to the user interface. For example, any parameter settings will have to be made as easy as possible, and modifying the DSM should also be possible with limited effort, to facilitate trying out different scenarios. At the same time, some information should be provided on the inner workings of the tool, so that users can evaluate the results. This includes a high-level view to the algorithms, and explanations on the action proposals.

It should also be possible to extend and modify the tool. Firstly, this study is the first attempt at developing the EA work to this direction. As such it is highly likely that e.g. new architecture views and metrics will need to be added as the tool is used. Secondly, beyond the initial aim of using the tool in the setting described in chapter 4.1.1, the best case is that the tool would find use also at other ABB sites, with their own interests for the architecture.

Firstly, there was a wide variety of analysis and simulation programs that utilize DSM as their primary element or offer it as an option. In this study they will be called

"dedicated software", as the DSM plays a central role in these applications. The benefit of dedicated software is that they often offer a user-friendly interface, and depending on the program, analysis tools for different business needs. Usually they also have a selection of algorithms available for sequencing, clustering and otherwise modifying the DSM, and metrics that can be calculated. They can also be efficient in calculations, as the focus on the DSM allows optimizing the code for this function.

The disadvantage on the other hand is that these programs do not usually offer extensibility. Many allow setting varying element and connections attributes, and thus creating different views, but for example implementing ABB-specific metrics is often not possible. Another drawback is that these programs are typically not free; therefore the question of cost and value should be considered.

Yet another drawback is that although the programs offer clustering and sequencing, most often the algorithm used for accomplishing these goals is not specified. There are a multitude of approaches for both goals, however, and as such these should be made clear to the user in order to assess, what kind of results the user should expect and how reliable they are.

For example, returning to the different clustering algorithms presented in chapter 4.4.3, the algorithms have quite different aims: those presented in (Thebeau, 2001) and (Yu et al., 2007) try to balance their solution between the number of connections outside clusters and cluster size, and therefore the algorithms have optimal solutions that they are trying to achieve. By contrast, (Zakarian, 2008) uses only the number of connections outside clusters as the objective function, and thus the optimal solution of the algorithm changes each time the algorithm is run, even if the underlying DSM is still the same. Now, if the user is not given information on this subject, or doesn't even know about potential differences, their ability to interpret the solution offered is severely hampered.

The second option was Microsoft Excel ("Excel" in the following text), the near-ubiquitous spreadsheet software. Excel offers programming capabilities in the form of Visual Basic for Applications (VBA), which allows functionality beyond that offered by Excel itself.

The first benefit is that implementing the tool in Excel enables a wide range of features that can be implemented; Excel does not limit the data used in any way, so different views to the architecture and ABB-specific metrics can be created fairly easily. In addition, the program housing the source data had a feature that allowed data to be exported into an Excel spreadsheet; with Excel, the source data could therefore be used with few adjustments. Lastly, Excel is widely adopted at ABB, which carries two benefits: firstly those using the final tool will be familiar with the software, at least to some extent, and secondly the program will also be available to everyone, with no extra costs.

On the downside, Excel is of course not designed for just DSMs. This means firstly that the user interface will on one hand be quite rigid, on the other a bit too free. As for the rigidity, for example moving an element within the DSM will mean cutting and pasting, not just dragging the element to a new place as in many dedicated programs. This means that using the tool will require the user to play by Excel's rules, which in

this case could be unintuitive. On the other hand, the freedom that using Excel brings means that preparing for errors is troublesome. For example, if the tool is to work, the DSM will have to be in a specific place on an Excel sheet, so that the code can refer to it. Run-time checking of the position of the DSM would be very hard and time-consuming to implement.

In addition, VBA is an interpreted language, which means that it is not compiled into machine language before running it. While this allows rather easy debugging and run-time viewing of the code, it makes running the code slower than when using a compiler. This could especially be a problem when running clustering algorithms: as they perform a large number of operations, even slight efficiency differences in running individual operations could mean differences of several hours in total time consumption. This will of course affect the usability of the tool.

The third alternative was MATLAB, a well-known analysis software product (MathWorks, n.d.). The benefits of MATLAB are firstly that it offers the same freedom in implementing features as Excel, as MATLAB has its own scripting language available. Secondly running code in MATLAB is much more efficient than running it in VBA. For example, (Börjesson and Hölttä-Otto, 2012) cite the time taken for one run the IGTA-Plus algorithm as 0,34 seconds using MATLAB. When implemented to VBA, the same algorithm takes roughly two seconds per run, which, while not impressive individually, means that time taken for 10000 runs is increased from about 50 minutes to 5,5 hours.

On the other hand, three drawbacks were identified to choosing MATLAB. Firstly, it is not nearly as well known to the target user group of the tool as Excel, and it might not even be consistently available for use due to software license availability. It can be safely assumed that those using the tool will have much more experience using Excel than MATLAB, which means that an Excel-based tool should meet fewer problems caused by user inexperience. Secondly, MATLAB does not offer even Excel's level of user interface. Therefore, for the purposes of this study, it is not a very good choice for testing scenarios or visualizing the DSM. Thirdly, the source data will in any case have to be brought in from Excel, which means that some kind of plug-in to read the DSM info would have to be implemented.

In the end, dedicated software was ruled out as an alternative, at least for the time being. A few programs were tested, but they usually offered unnecessary functionalities and did not enable customizations. Thus, they did not seem to offer enough value for money at this point.

As for the question of whether to use Excel or MATLAB, as explained before, Excel was chosen because it is more familiar to the probable user group of the tool, it offers better capabilities in visualization, and the source data are in Excel form. In addition, ABB has Excel in virtually every workstation, which is also an advantage over MATLAB.

An interesting way to combine both approaches would be to use MATLAB for the computing-intensive calculations, such as running the clustering algorithm, and then

bringing the results back to Excel for analysis. Combining the benefits of both programs seems a like potential ways of further developing the tool.

5.3 Initial DSM

The starting point for the study is shown in Figure 5-1. This represents the data as produced by the printing method of MoGeApps, based on the application and interface catalogs. Some minor modifications have been made, as explained below.

The elements of the DSM represent individual applications in use. In Figure 5-1, application names have been replaced with numbers, as they do not affect the outcome of this thesis. This approach will also be used in the rest of this thesis. Another modification is that applications that were present in the application catalogue but that did not have any in- or outbound interfaces have been removed. Although these can be interest-

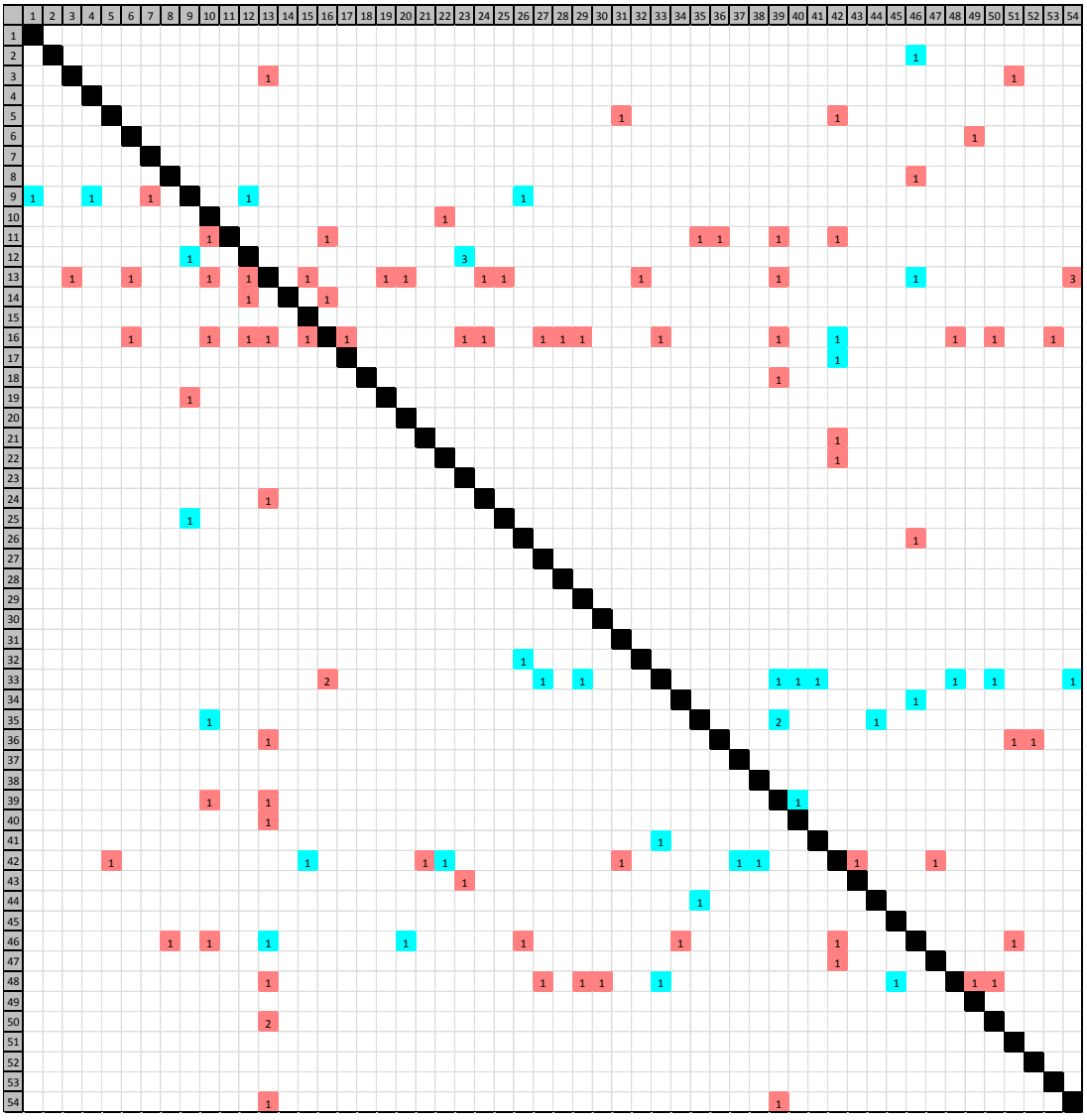


Figure 5-1: Initial DSM (25.10.2013)

ing when applying the tool to practical use, they do not bring much value to this thesis, as the approaches discussed here have almost exclusively to do with the interfaces.

The marks in the DSM represent the interfaces between applications, as recorded in the interface catalogue; in other words one program provides information to another. To reiterate, at ABB the DSM is formed so that source applications are in the rows and target applications in the columns. Thus, looking at the Figure, if there is for example a mark in row 9, column 1, it means that application 9 provides information to application 1.

The two different background colors of the marks in the DSM represent different technologies the programs use in transmissions. The red color means that there is a direct connection either between the programs or databases that the programs operate. The cyan color on the other hand represents a file-based transmission: either the source program creates a file that the target program then reads, or both programs utilize a shared file. These represent initial areas of interest of the project.

The values in the marks represent the number of methods used in transmitting information between each pair of applications. For example, the interface from application 12 to application 23 has the value 3, which means that the transmission of information utilizes three methods. Another value that could be extracted from MoGeApps was the number of information fields that are transmitted between the applications. Again, these represent initial interests.

This was initial setting of building the tool. Specific development steps will be discussed in following chapters.

5.4 Depicting interconnections

The main question concerning the interconnections in the DSM was what the interactions should represent. Related to this was the question of whether to use a binary or numerical DSM. As explained above, initially the source data provided two ways of presenting a numerical DSM, which could of course be reduced to a binary DSM. While it might feel more informative to use a numerical DSM, it places requirements on the source data not necessary when using a binary DSM.

If data represents the quantity of something instead of, for example, a category, the quantity should naturally carry a meaning; a higher number in this type of numerical DSM should represent a stronger or more important connection. As discussed in chapter 3.3, in literature the primary interest with regard to the interconnections is the probability of change propagation. This is measured by various proxy values, with a higher number representing a greater probability. This places demands on the granularity of information in the DSM. If this type of numerical DSM is used, then on the widely adopted Stevens scales of measurement (Stevens, 1946), the scale used for presenting the values should be a ratio scale. A ratio scale is the most permissive in terms of operations, allowing multiplication and division in addition to all other operations. These are

necessary for e.g. matrix multiplication, which in turn is used in many DSM operations. Normalizing the values does not remove this problem, if the initial data are not suitable.

The literature examined is mostly focused on analyzing single applications, with the values in the DSM related to the probability of changes propagating to other elements, as discussed in chapter 3.3. Even the research done in EA setting the marks most often represent quite practical information, for example in (Lagerström et al., 2013b). In the context of this study, however, it was not self-evident if the same approaches could be used, as the marks represent more than just syntactical connections between applications. Firstly, the exact nature of the information provided from one application to another is not present in the DSM, although it is recorded in the interface catalogue. This means that it was not possible to discern to criticality of the information being passed with regard to the functioning of the order-delivery process. By extension, if the importance of a single-method interconnection could not be seen in the DSM, the same applies for multi-method interconnections. With regard to the other value available, the number of transmitted fields, again, the volume of transmitted information was not a viable proxy for the importance of the information.

Additionally, the information transmitted between programs does not necessarily belong to the same context. Several of the elements represent large applications with diverse roles in the process, the best example being the ERP system in use at ABB. To portray this kind of sizeable application consisting of several modules as just a single element necessarily hides information, as different parts of these programs could handle highly different data. This is a clear distinction between analyzing architectures in the context of a single application and an EIS, as the connections in a single application usually concern lower-level information used to run the application.

Secondly, the number of the methods or fields transmitted was not considered to reflect the complexity of the interconnection, the technology used in the transmission was evaluated as more important. For example, a transmission using a connection to an SQL database could be easy to modify, and if either the source or target application was changed or replaced, the architect could be quite certain that the new program would also be able to perform a similar connection, due to the wide acceptance of SQL. Whether the connection then uses one or three SQL connections does not make a large difference.

On the other hand, a direct connection between two programs could be more difficult. Even if the interfaces between the applications were well documented, the new application could be a third-party product and therefore might not be able to fulfill the same interface. Again, this is a major difference between the domains of single applications and information systems: the size and informational complexity of interconnections in an EIS do not allow a similar plug-and-play approach to building the system as when creating a single application. There are ways of achieving similar objectives in an EIS context, such as using special integration software, but implementing those are a great deal more difficult than modifying the interfaces of a single application.

Therefore, the initial numbering scheme was not considered useful. As explained above, the transmission technology used was identified as a major source of interconnection complexity. There was initial interest in applying scales presented by (Offutt et al., 1993) and (Xia, 2000) as an alternative numbering scheme in order to make interconnection values comparable to an extent. Unfortunately, this was quickly found infeasible. Firstly, the number of different technologies in use for the transmissions was quite large, and drawing comparisons from these to the data types presented in these sources seemed like too much of a stretch to be useful in the context of this study. Secondly, neither measurement scheme produces results on a ratio scale; at worst the results are only ordinal in nature. Thus the schemes would not reflect reality any more than the initial scales. In addition, no literature utilizing similar scales in the EA domain could be found. In total, it was decided that using these schemes was not worth the effort.

However, because of the importance of the transmission technology, it was decided that they would be useful to be represented in the DSM. Because no suitable ways of calculating the interconnection complexity could be found, the easiest alternative to implement was that the users of the software could evaluate the complexity themselves. As this introduces subjectivity to the results, it was though best kept separate from the code and given to the user to define as parameters. However, using just cell values would not be the best way of visualizing the architecture; that goal would be better achieved by different coloring of the marks, an example of which can be seen in Figure 5-1.

Thus, the transmission technologies were incorporated to the DSM in two ways. After the tool has read the transmission technologies from the interface catalogue, the user can define formatting for each type of transmission technology. This aids in visualizing the specific technologies. In addition, a comment containing the corresponding technology is added to each cell for ease of use. The user can also input any value for the technology, which is then input to the mark in question. This way the user can define values for the complexity of each transmission type if so desired. As the values are

Technology	Formatting
COM	1
DDE	1
File Transfer	1
FTP	1
HTTP	1
LEI	1
Shared File	1
SQL Server Connection	1
TCP	1
Web Service	1
...	

Figure 5-2: Part of table containing formatting for transmission technologies

present in the DSM, they will subsequently be used in any operation involving the DSM values.

A section of the coloring scheme used is presented in Figure 5-2. The architecture with example formatting added is shown in Figure 5-3. For examples sake, each technology has been given a value of 1; this would make the DSM binary.

The tool was also built so that the user can substitute practically any connection attribute to the place of the transmission technology. This allows additional flexibility in creating new views to the architecture in the future.

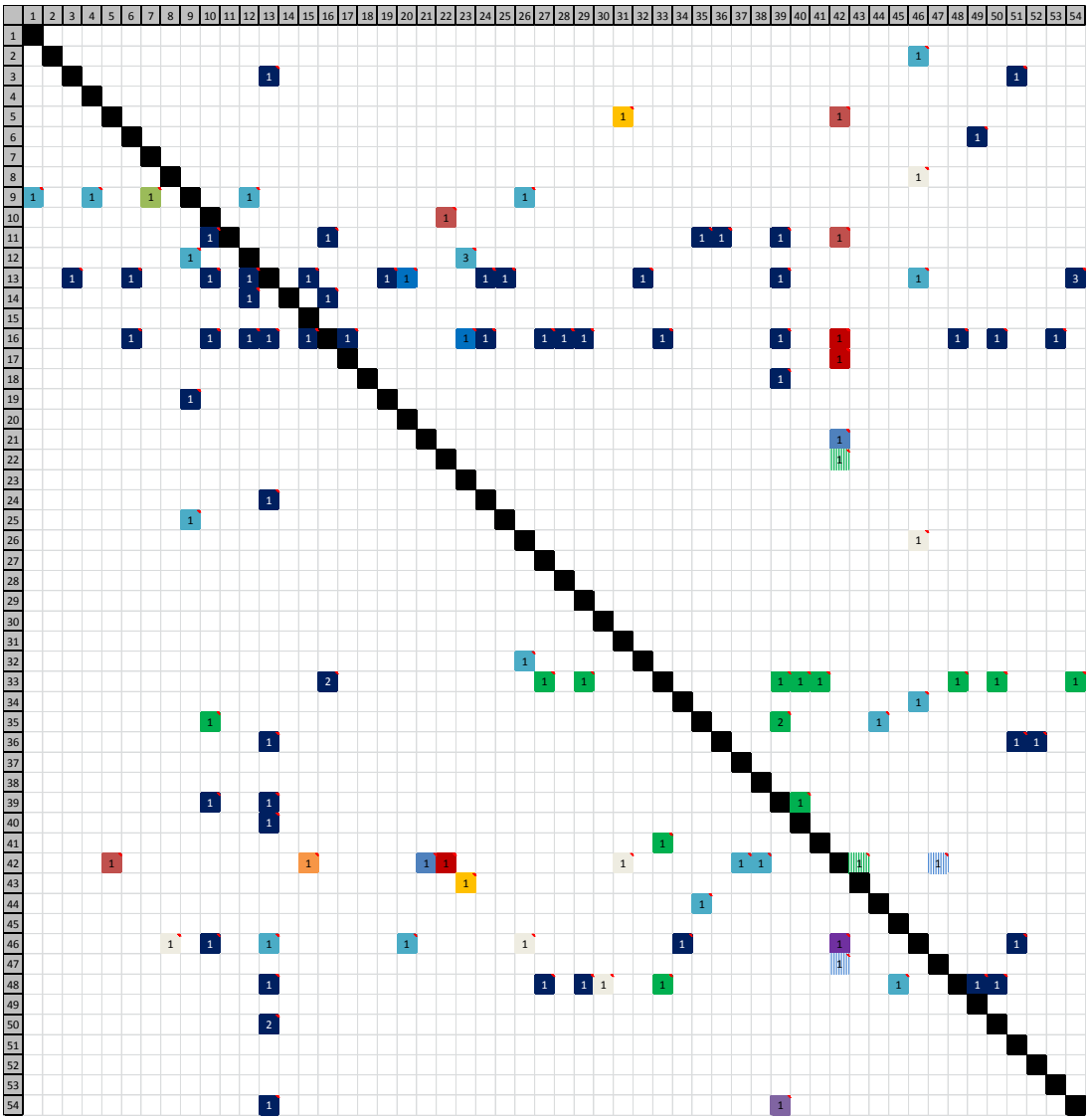


Figure 5-3: DSM with formatting added to marks

6 BUSINESS MODULES AND INTERFACE SUGGESTIONS

This chapter discusses dividing the architecture into business modules and creating interface suggestions between them. However, to this end the application visibilities are also used, so they are discussed first. Afterwards, the method of dividing the DSM by business module is presented, with the principle behind the interface suggestions coming last.

6.1 Application visibility

As discussed, one of the views deemed useful was application visibility and indirect dependencies between applications. This has been used in literature for various ends, most notably for identifying core and peripheral components and cyclic groups of a system. Implementing this is discussed next.

In more general terms, the visibility matrix of a DSM is equal to the transitive closure of the matrix, and there are several approaches for calculating it (see for example Nuutila, 1995); the method of matrix multiplication was already discussed in chapter 3.4.2. In this study, the method chosen was the Floyd-Warshall algorithm, which is well-known and efficient. The Floyd-Warshall algorithm calculates in one run the shortest paths between all element pairs in the matrix. As a downside to its efficiency it does not allow examining of the paths themselves, only the path lengths are recorded. Other path-searching algorithm such as Dijkstra's algotihm can be used if the paths are of interest. However, in the context of this thesis and based on the literature, the visibilities were considered more interesting.

For identifying cyclic groups of elements, the method described in (Baldwin et al., 2013) was used. This was chosen for its simplicity and due to the fact that it has been utilized in research closely related to the subject of this study (Lagerström et al., 2013a, 2013b). In short, the method consists of the following steps:

- Calculate the visibility matrix
- Sort the visibility matrix first by fan-in visibility (FIV) in descending order, then by fan-out visibility (FOV) in ascending order
- Set each group of elements with the same FIV and FOV that are greater than one into same cyclic group

An interesting observation is that this method of identifying the cyclic groups is largely the same as the one described by (Warfield, 1973) for dividing a matrix into hierarchical

levels. Put simply, this means all connections are arranged to one side of the DSM diagonal, or into blocks on the diagonal. This, in turn, corresponds with sequencing the DSM. This will be returned to in later chapters.

As for the core and peripheral elements, (Baldwin et al., 2013) base their classification on the size of cyclic groups detected in the previous step, relative to each other and to the size of the DSM. The important dividers presented in their study are in sequential order: does the largest cyclic group contain more that 4% of system elements; is the largest cyclic group more than 1,5 times larger than the second largest cyclic group; and does the largest cyclic group contain more than 6% of system elements.

However, it must be noted that these values were used in the domain of a single software system, with a much larger number of elements. Therefore using these exact values in the EIS domain should not be taken as a given. Especially the percentages of system elements in the largest cyclic group seem quite low: for the architecture of a software product of, say, a thousand elements, 4% would mean 40 elements, which would stand out in the DSM. However, for the architecture being studied in this thesis, with 54 elements, 4% means approximately two elements and 6% about three. Although groups of this size would be visible in the DSM, a group of three elements hardly seems worth calling the core of the architecture. Thus, the exact values were not taken as a strict rule, but more as a guideline to classifying the architecture.

If the architecture shows of signs of containing a core group of elements, the rest of the elements are classified by their visibility values relative to the core. These visibility groups counting from the top of the DSM are as follows: shared (FIV higher than core, FOV lower than core), core, periphery (FIV lower, FOV lower), and control (FIV lower, FOV higher). The names come from the presumed function of the groups. The shared group represents e.g. library functions that provide information and services to components, but receive them from few. The control components are the opposite, and can represent e.g. user interface functions that primarily work by using other functions. The periphery group has the same meaning as in literature concerning core components in general, and they are only weakly connected to the rest of the system.

The visibility matrix that has been organized using the approach defined above is presented in Figure 6-1. The marks with a value greater than zero have a blue background. Cyclic groups have an orange border and the visibility groups described above have a red border. A value of zero means that there is no path between the two elements; other values signify the number of steps on the shortest path between the elements.

Examination of the matrix shows a clear core group consisting of 15 applications. This represents roughly 28% of the system, and the group is three times the size of the next largest cyclic group. Thus, even if one were to disagree on the exact divider values used by (Baldwin et al., 2013), the largest cyclic group is quite clear in its dominance of the architecture.

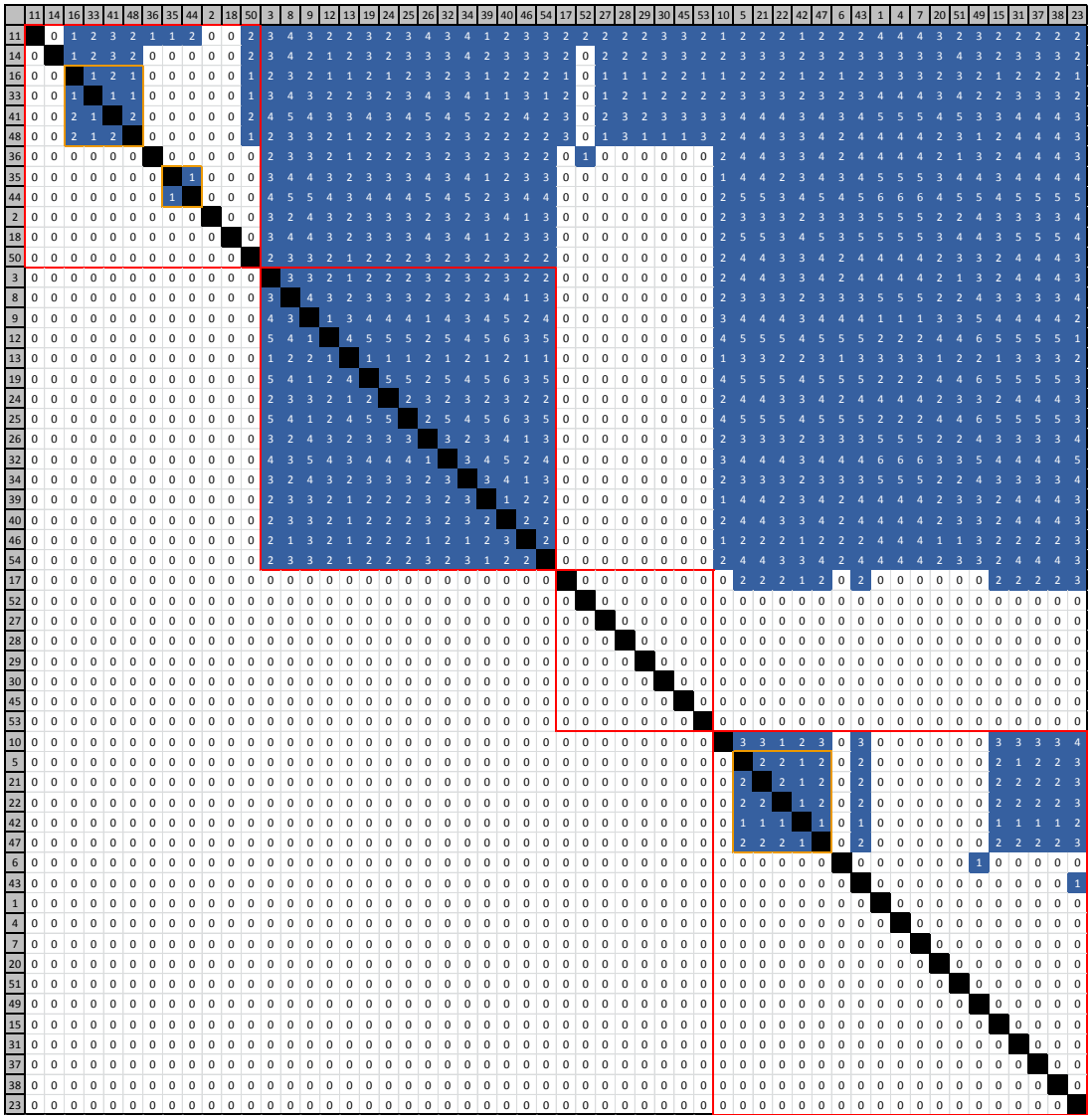


Figure 6-1: Visibility matrix of the DSM. Cell values indicate the length of the shortest path between two elements, a value of zero means the absence of a path. Cyclic groups have been marked with an orange border and component classes with a red border. The core, the second group counting from the top, is at the same time the largest cyclic group.

Organizing the DSM and identifying the cyclic groups provided some interesting information on the architecture. Firstly, the positions of the elements in the visibility matrix correspond quite well with their positions relative to the order-delivery process. Elements before the core are mostly applications used in the sales phase. These include product configurators and other applications used in defining the specifications of the product that is to be produced. On the other hand, applications in the last group were applications used in the later phases of the process. These consist mostly of viewer-type applications, for example, viewing the production structure of the design. This is to be expected, as by the end of the process few changes should be made. Thus, it is logical that these applications receive much more information than they produce.

Secondly, the core consists predominantly of a group of applications used in the design phases of the order-delivery process, and thus forms a quite easily understandable block of applications. This result is important in order to visualize the problems that could arise if making changes to this group. As an example that came up during the conversations regarding the architecture, the higher management of the business unit often push for certain applications to be adopted at all sites of the business unit, in order to narrow the application portfolio. The aim is to focus resources into support and development of a smaller number of applications, and also to facilitate the exchange of data between sites; these are also the aims of this project. A few of the applications in the core group are these kinds of applications sometimes suggested for wider adoption. However, looking at the core application group, it can easily be seen that picking just some applications from the group will be problematic; as they receive information from and produce it to many all other programs in the group, then in order to implement an application at another site, it would require a similar "infrastructure" of other applications with it. The same applies to the other cyclic groups.

As for dividing the other elements into the four visibility groups, this approach was not considered as useful. Firstly, as the applications are linked to a certain part of the order-delivery process, it is logical that the earlier an application is in the process, the higher FIV and lower FOV it has. Secondly, the groupings were problematic because, as discussed in chapter 5.4, the information flowing from one application to another is typically not the same information that the receiving application then passes forward. Therefore, looking at Figure 5-4, it might not be correct to present an indirect connection between two applications that are very far apart in the DSM; when the latter application is reached, the information will have undergone many changes in both type and content. For this reason, using the visibility DSM in ABB's context could be misleading. The situation would be different if the connections in the DSM were semantically closer to the syntactical connections used in literature. However, in ABB's case, analyzing the visibility matrix also requires knowledge of what information is passed in reality.

Thus, a DSM sequenced with the visibility matrix was considered more useful for ABB's purposes. This is presented in Figure 6-2. The information content is largely the same as in Figure 6-1, but only the direct connections are shown, with the purpose of keeping the focus in the direct connections. In addition, the cyclic groups identified are shown. These were considered useful, especially if the knowledge of the cyclic groups can be combined with knowledge of what elements are within them. The bordered groups with no elements in them represent groups with the same visibility. These are bordered to remind the user that the sequencing does not take a stand on their mutual ordering.

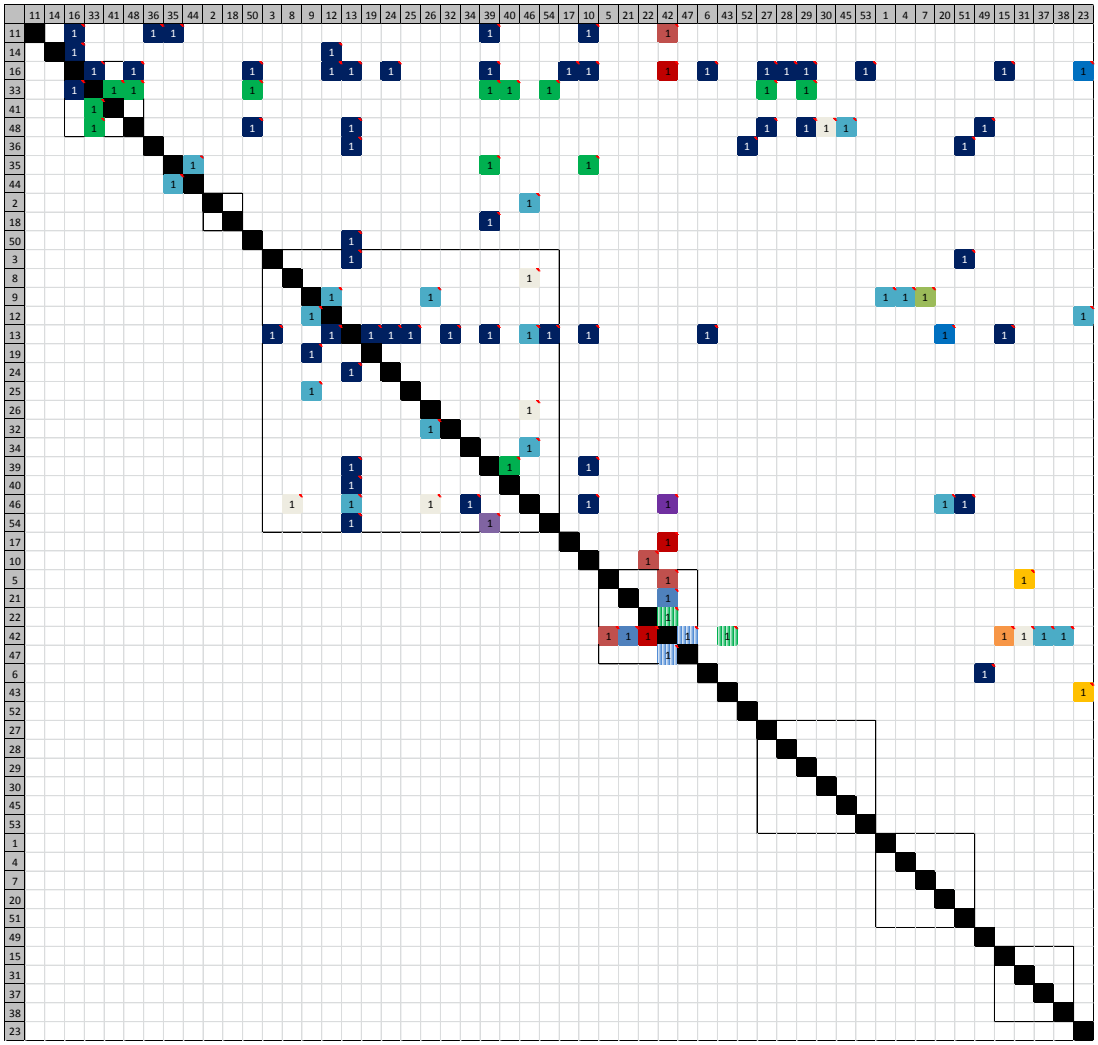


Figure 6-2: DSM sequenced with the same approach as with the visibility matrix.

6.2 Incorporating the business modules to the DSM

At first the source data did not include information on which business module each application belonged to, so the first step was to collect this information. To this end, new data fields were created to the source data for the business modules and for the level of ownership of the application.

The data for each application relevant to the scope of the project was input by the BU IS manager and another employee who had been involved in the project from an early date. Another alternative would have been to perform a second round of data collection from application owners. However, this was decided against for two reasons. Firstly, it was felt that the data could be input accurately enough by the two persons mentioned, and distributing the data input task would have taken unnecessarily long. This thesis aims at creating a tool which can be used after each round of changes to the architecture; therefore, the source data can be clarified at a later date, after the tool has been in use for some time and the process of using it has been tested. Secondly, it can be

argued that it is better that the personnel making decisions on the direction of the architecture make the decision of the module division. This way, the source data will represent the vision of the personnel who will be using the tool.

After collecting the data on the business modules, modifications were made in the database in order to record the data and to export them alongside the DSM. The data were added to the DSM as attributes of the elements, to the left of the DSM.

As discussed, the tool needed to be able to produce two different recommendations for how to connect the business modules to each other: one with connections from module to module, and one using a bus structure. In practice this was implemented so that the user could sort the DSM based on element attributes of their choosing, the most interesting for the purposes of this study being of course the business module of each element. If the user would erase the attribute information for some element, then that element would be set as a bus element, and would be placed to the lower right corner of the DSM.

In addition, the initial values for the business module attributes were simply the names of the initial process phases shown in Figure 4-2. Due to how Excel sorts values, this would lead to the modules appearing in the wrong order in the DSM; mainly, "sales" would appear second to last instead of being first. For this reason, an additional functionality was implemented that allowed replacing attribute values with other values of the user's choosing. This way, the names could be replaced with e.g. numbers and order of the modules would be made correct.

6.3 Interface suggestions for the architecture

The next step was to decide how to produce interface suggestions from the DSM and how to portray them. Separate suggestions would have to be created for both module-to-module and bus-based suggestion, as discussed in chapter 4.1.2.

The problem will be approached in separate steps. Firstly, let us consider what it means for a system's architecture to be partially or fully hierarchical; this has been discussed in e.g. in (Warfield, 1973) and more recently in (Sangal et al., 2005), with the names "layered" and "fully layered system". The word hierarchy as defined by Merriam-Webster means "a system in which people or things are placed in a series of levels with different importance or status" (Merriam-Webster, n.d.). What this means in the DSM context is best elaborated with an example, which is presented in Figure 6-3, adapted from (Warfield, 1973, p. 446). The matrices in the figure have been transposed to comply with the order of reading the DSM used in this thesis; the information content is unchanged.

The left matrix in the figure is the initial situation for the hierarchy analysis. The right one is the same matrix that has been reorganized so that hierarchical levels can be made visible. A hierarchy in this case means a DSM that has been divided so that feedback is only allowed within blocks that are symmetrical around the diagonal, not between the blocks. Therefore, the sequenced DSM shown in Figure 6-2 is also a hierar-

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	1	1	0	0	1	1	1	0	1	1	1	0	0
2	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	0	0	1	0	0	0	1	1	0	0	1	1	0	0	0
4	0	0	1	1	0	0	1	1	1	0	1	1	1	0	0
5	0	0	1	1	1	0	1	1	1	1	1	1	1	1	1
6	0	0	0	0	0	1	0	0	0	1	0	0	0	1	1
7	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0
8	0	0	1	0	0	0	1	1	0	0	1	1	0	0	0
9	0	0	1	1	0	0	1	1	1	0	1	1	1	0	0
10	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1
11	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

	2	1	5	6	10	9	4	14	15	13	3	8	7	11	12
2	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	1	0	0	0	1	1	0	0	1	1	1	1	1	1
5	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1
6	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0
10	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0
9	0	0	0	0	0	1	1	0	0	1	1	1	1	1	1
4	0	0	0	0	0	1	1	0	0	1	1	1	1	1	1
14	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
8	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
7	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
11	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 6-3: Example of dividing a DSM to hierarchical levels, with connection areas of a fully hierarchical system marked on the right (adapted from Warfield, 1973)

chical division, with the bordered groups of elements representing levels of hierarchy. Comparing with the dictionary definition of hierarchy, the elements have been divided into groups based on their fan-in visibility, as a measure of their "importance" or "status" The aim is to move all connections between elements to the upper right segment of the DSM. If a bordered group of elements has connections within it, it means that the elements form a cyclic group; if there are no connections within the border, it simply means that the elements are on the same level of the hierarchy, and as such the sequencing does not take a stand on the order of the elements within it. This information might not feel as useful, but it was decided that it should be included as a reminder of the limitations of the sequencing.

In the words of (Sangal et al., 2005), this type of system is "layered": each layer, or level of the hierarchy, depends only on the layers below it. This type of analysis aims at eliminating feedback connections, which are seen as harmful to the development of the system.

Another step is to try to make the system fully hierarchical, or fully layered. This means that each level of the hierarchy is only connected to the immediately neighboring levels. In Figure 6-3, these are presented as the areas between the element groups that have a dashed border. Making the system fully hierarchical is an additional measure of limiting connections between hierarchy levels. Whether this is beneficial depends on the context. The benefits of eliminating feedback connections are easier to see as they pose an immediate risk of change propagation. Aiming for a fully hierarchical system does not eliminate any feedback connections, but limits the extent of the feedforward connections.

As for the context of this thesis, a fully hierarchical system seemed to be the most sensible. If the DSM is sorted based on the business modules, then they represent the initial levels of the hierarchy. Allowing connections from, say, straight from sales to service would not be in line with the goal of increasing the system's flexibility. Each connection from a business module to other modules than those directly adjacent to it

will make it harder for different sites to make their own decisions on how implement the applications in each phase of the order-delivery process, as they create dependencies to design decisions beyond those strictly required for the functioning of the process.

With this goal in mind, the next phase was to implement looking for interfaces between the modules. This was divided into two sub-goals. Firstly, the element providing the information to the next business should represent as much of the information in its business module. This is as close as possible to the initial goal of just a single connection relaying all information between the business modules, using just the existing connections in the DSM. Secondly, the element in the module receiving the information should be as early in the information flow of its module. Again, this way the receiving element would be theoretically able to provide information to as much of its module as possible.

To find out the place of each element in the information flow of its module, a two-level hierarchical division was implemented. Firstly, as discussed, the DSM is sorted based on the business module attribute of each element. Secondly, the elements in each business module are sequenced based on their visibility within the business module. This brings to light the logical order of the elements within each business module. After sequencing, the position of each element represents the place of the element in the information flow of the business module, and can then be used as a basis for searching for interface applications.

After organizing the DSM as presented above, comes the actual phase of looking for interfaces. Again, this was implemented in two levels. The process is given two business modules as inputs; let us call these the business module providing information, or "providing module", and the business module receiving information, or the "receiving module". First, the elements in the providing module are looped through, starting from the last one. In this first phase, for each element it is first checked if the element receives information from other elements in the module; due to the effects of indirect connections and how the elements are sorted based on them, it is possible for an element to be quite late in the architecture, while still not having any direct inbound connections.

This check reflects the goal of finding an element that "condenses" the information in the business module. If an element is late in the hierarchy but receives no information from other elements in the module, it does not seem ideal as an interface; it could be largely separate from the main information flow within the module. Of course, it could then be asked why it has been included in the business module at all. However, that question has to do with the quality of the source data, which is a concern separate from this study.

If an element receives information from other elements in its module, then it is checked if the element has connections to the receiving module. Let us consider first the case that it does not. If there are still elements to check in the providing module, then the process moves on to the next element in the providing module. If this element was the last one checked, then a second round of searching is initiated again from the last element in the providing module. This time the requirement of getting input from other

elements in the module is relaxed; the first element that provides information to the receiving module is considered fit for being the interface application. If this second round of searching does not reveal any suitable applications in the providing module, then there are no possible interfaces between the two modules, and the search moves on to the next pair of business modules.

Let us now return to the situation that an application receives information from other elements in the providing module and provides information to an application in the receiving module. At this point the search process limits the search to the hierarchical level of the identified providing application and receiving application. As discussed, the sequencing method used does not sort elements within hierarchical levels; therefore, if we consider the information flow within a module, applications on the same hierarchical level should have the same chance of being selected as the interface application. With the appropriate hierarchical level identified in both the providing and receiving module, all connections between applications in these groups are added into a list for consideration.

If at this point there is more than one potential pair of elements to serve as the interface, several methods for choosing one pair over the others could be used. As discussed in chapter 4.3, two factors that were hypothesized as having an effect on the suitability of the interfaces were the visibility group of the elements and the level of ownership. However, neither of these was implemented in the end, although the business level was brought to the DSM as an application attribute.

The reason for these being left out was that no theory could be found to definitely put some attribute values above others, and therefore the choice would have to be based on some kind of subjective valuation scheme. This in turn was felt as unnecessary; as discussed in chapter 1, the aim for the tool is to create architectural suggestions based on theory, and these are then modified for practical uses. Bringing in subjectivity already at this point was felt as redundant. Also, time constraints limited efforts in developing the tool to the issues felt as being most important, and as such this feature was left out.

As it stands, then, the pair of applications chosen as the interface suggestion is simply the first pair in the list, i.e., the last application in the providing group and the first application in the receiving group. This pair is then marked into the DSM, and the analysis moves on to the next pair of business modules.

In Figure 6-4 is presented the way the tool visualizes the development suggestion for the architecture, built around moving information from module to module. The business modules are surrounded with a thick border; they have also been organized so that they are in the order of the order-delivery process. Within the business modules the application elements have been sequenced as discussed. Also, as in Figure 6-2, the applications on the same hierarchical level have been surrounded with a thin border.

Other visualized sections of the DSM are the acceptable and forbidden interface areas between modules. As discussed, a fully hierarchical architecture was considered as the way to go. Therefore the acceptable area for interfaces is the area to the right of

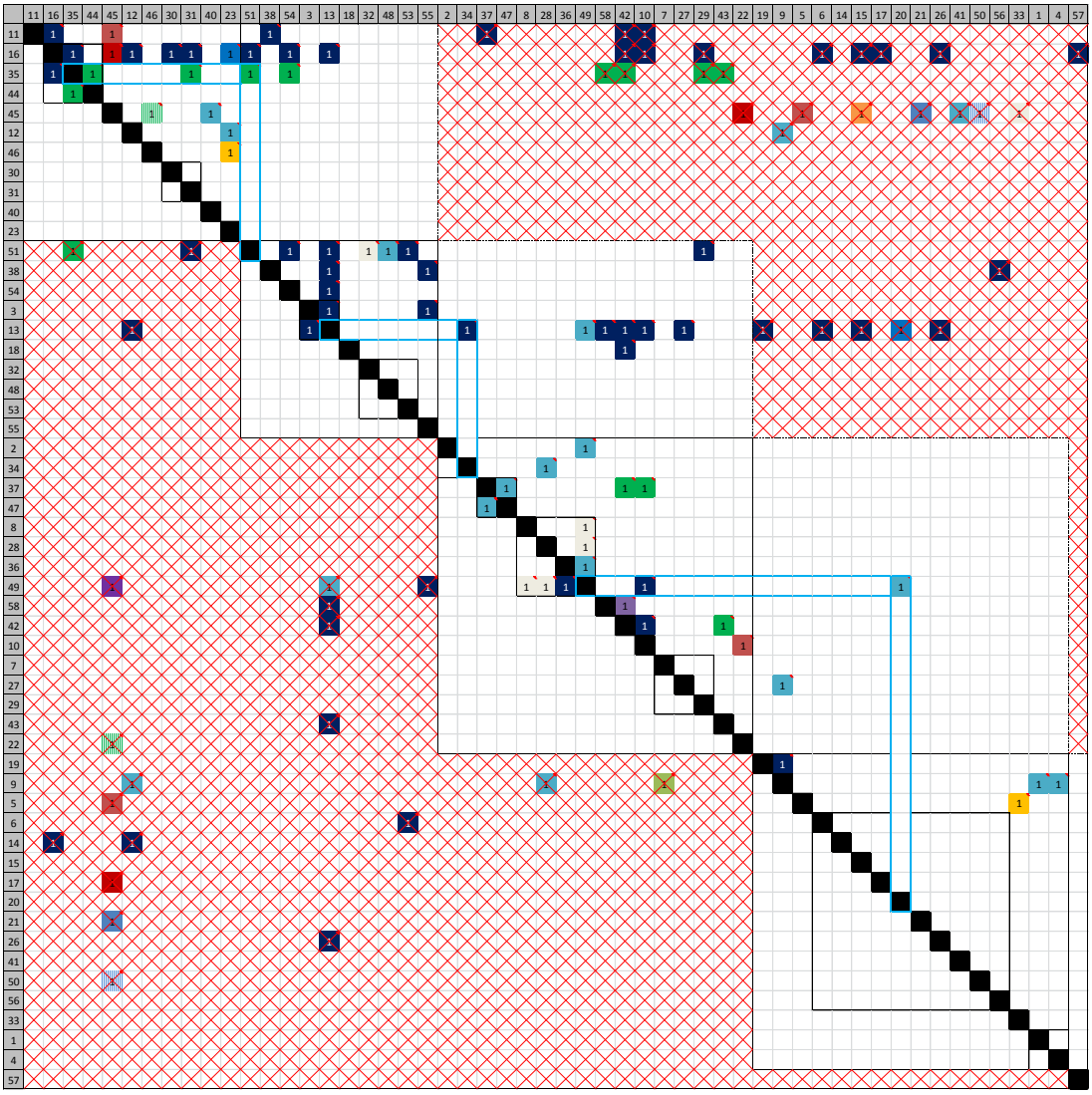


Figure 6-4: Interface suggestions for moving information from module to module, with elements divided into business modules and sequenced within the modules

the providing module and above the receiving module; these are shown in the figure with a dashed border. All other connection areas are forbidden with regard to the fully hierarchical setup, and these areas of the DSM are visualized with red crosses over the cells. The last component is the interface suggestion itself which having been found previously is presented with a blue border.

The architecture suggestion based on the enterprise bus is created to a large extent in the same fashion as the module-to-module suggestion; the most important difference is that the connection from the providing module to the receiving module is sought using the bus defined for the architecture. So, in the first round, if an application in the providing module has inbound connections within its module, a connection is looked for between that element and the bus elements. If such a connection is found, another connection is the looked for between that bus element and the elements in the receiving module. If such a connection can be found, then all the connections between the two hierarchical levels are analyzed, as with the module-to-module suggestion. If a connection

that uses the bus cannot be formed between the two analyzed business modules, then the tool attempts to form the connection in the same fashion as in the module-to-module suggestion; if a connection cannot be found in this fashion either, the tool moves on to the next pair of business modules. After going through each pair of modules, the user is notified of these failings to find interfaces, with an error message of the type shown in Figure 6-5 below. “No bus connection” means that an interface between the modules could not be formed using the bus. “No interface” means that no interface could be formed at all.

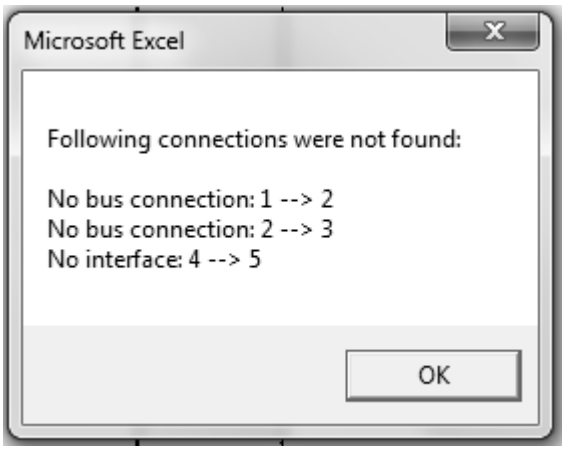


Figure 6-5: Example of error message provided by the tool to show failures in the interface search

Figure 6-6 pictures the architecture suggestion created by the tool that uses the bus. In this example, elements 15, 16 and 42 have been set as bus elements by removing the value from their cells in the business modules information column. Afterwards, they have been sequenced as the other modules, so they are not in the original order. As can be seen, interfaces that use the bus have been found linking each pair of business modules. As discussed previously, the architecture built around the bus presents a very different view to the architecture. With the tool, the architect can try out both approaches to create alternative scenarios.

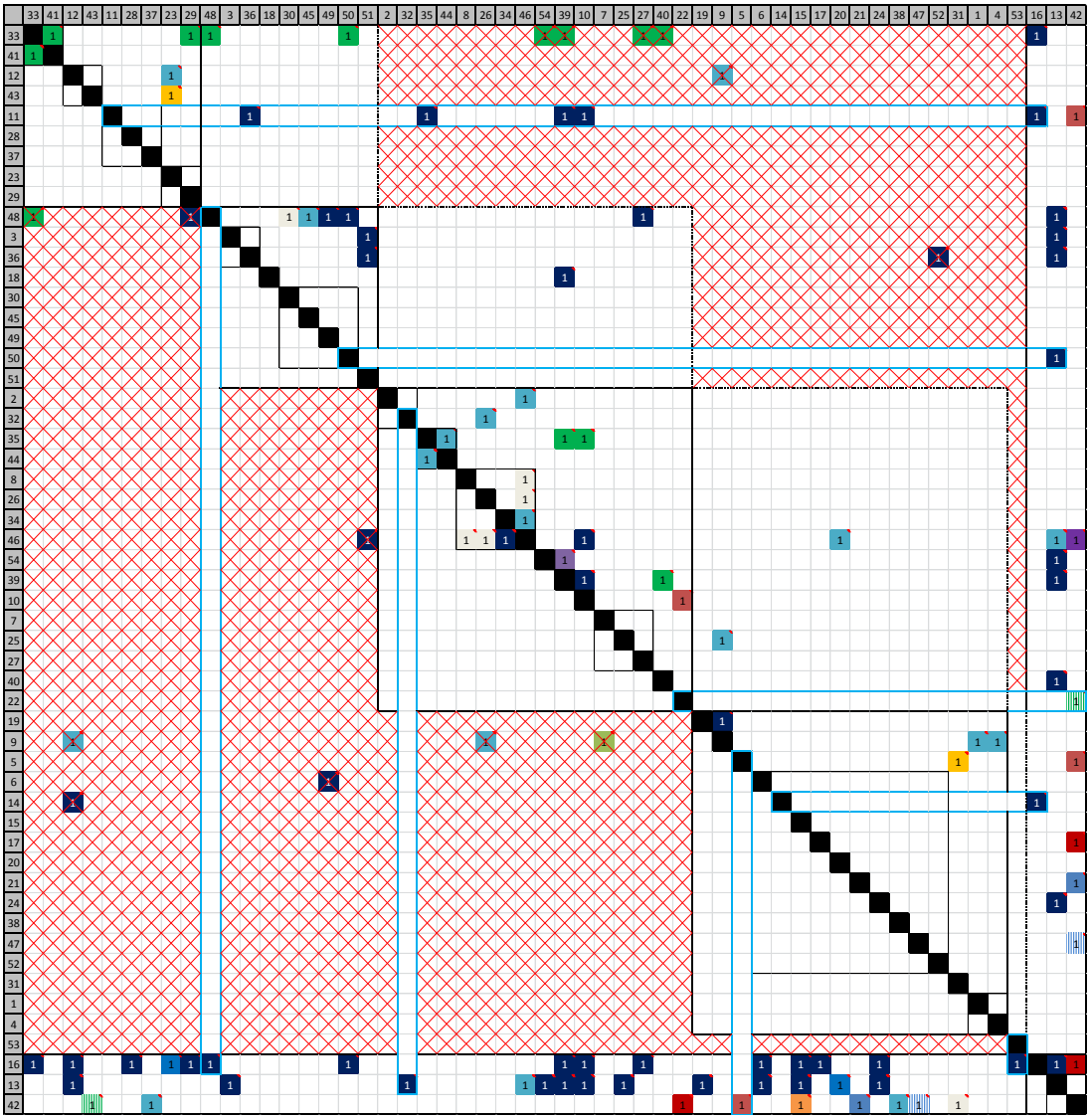


Figure 6-6: DSM with interface suggestions built around an enterprise bus of three elements

7 AUTOMATIC CLUSTERING

This chapter discusses implementing automatic clustering. Firstly, as discussed in chapter 4.4.3, a choice has to be made regarding what clustering algorithm will be implemented. Secondly, after choosing the algorithm, there are choices to be made in how to implement it. These will be discussed next.

7.1 Choosing clustering method

The three clustering algorithms presented in chapter 4.4.3 formed the initial pool for making the choice. The algorithms were (Zakarian, 2008), (Yu et al., 2007) and (Börjesson and Hölttä-Otto, 2012).

Zakarian's algorithm was removed as a choice quite early. As discussed, it does not provide a single solution that is considered best; instead the best solution changes with each iteration of the algorithm. Thus it was considered hard to understand by the future users of the DSM tool. In addition, the algorithm has not appeared in literature, and so no support can be found for its use.

The algorithm by Yu et al. was considered next. Their approach was considered promising, for two reasons. Firstly, their objective function is claimed to be capable of solving many problems present in other widely used clustering algorithms, most notably the wide-spread (Thebeau, 2001). The problems are presented in (Sharman and Yassine, 2004). Typical clustering algorithms are path-dependent, meaning that the solution presented will depend on the choices that the algorithm has made along the way. Therefore, if an element has equal grounds to be set into several clusters, the cluster it is set to will affect the rest of the running of the algorithm. In addition, typical clustering algorithms are claimed to struggle when presenting three-dimensional architectures. This is also a problem of the DSM in general, as discussed in chapter 3.2.3.

The algorithm by Yu et al. attempts to solve these problems by allowing an element to belong to several clusters simultaneously. It should be noted that the algorithm by (Thebeau, 2001) also allows multi-cluster allocation of elements; however, in Thebeau's approach, the elements are duplicated for each cluster that the element belongs to. This is done in order to artificially increase the size of the clusters, thus penalizing multi-cluster allocation somewhat. As a downside multiplication of the elements naturally clutters the DSM and could make it harder to comprehend. The approach by Yu et al. does not require replicating the elements in order to affect the value of the objective function.

The second reason the approach by Yu et al. seems promising is that it is claimed to be able to identify bus-like structures in the DSM. Thus, identifying bus elements in the

architecture could be made less subjective and let the algorithm decide. This claim has also been backed in (Hölttä-Otto et al., 2012). This is tempting as identifying bus elements has been done in various ways in the literature, and so presenting the best alternative for adoption at ABB is difficult.

For these reasons, the clustering algorithm by Yu et al. was chosen as the first attempted implementation of clustering. Implementation was attempted in the same configuration as in (Yu et al., 2007), meaning that both the objective function and the algorithm to try out different solutions were the same; Yu et al. utilized a genetic algorithm.

Sadly, this approach ran into difficulties quite quickly. These problems are discussed next. First of all, genetic algorithms are quite difficult to implement. To reiterate, the coarse functioning principle in genetic algorithms is that the prospective solutions are first coded into "chromosomes" consisting of "genes", in other words, arrays of logical values indicating whether a certain element is a part of the solution or not. In the approach of (Yu et al., 2007), the length of the chromosome is set as the maximum number of clusters, which is given as a parameter, multiplied by the number of elements in the DSM. The clusters can then be extracted from the chromosome by dividing it into sections the same length as the number of elements in the DSM. Thus, in the chromosome, a "true" logical value means that the current element is part of the current cluster.

After the prospective solution has been coded as a chromosome, the chromosomes are then crossed over, simulating reproduction in organic beings. What values are switched in the chromosomes must be given as a parameter. The pool of chromosomes chosen for reproduction is based on the objective function, although some randomness is usually also introduced, in order to prevent the algorithm from getting stuck on local optimal values, instead of the global optimum. After reproduction, some chromosomes usually undergo mutation, in which some values in the chromosome are changed to the opposite value, i.e., from true to false or vice versa.

However, here lies the difficulty of genetic algorithms: there are a great number of different methods to implementing the reproduction, and without extensive testing it is difficult to say which method is the best. For example, in the approach by Yu et al., when mixing the chromosomes, uniform crossover is used. This means that when performing the crossover, the probability of crossover is evaluated separately for each gene in the pair of chromosomes chosen for reproduction. An oft-used alternative method is the single-point crossover, in which a single point in the chromosome is chosen at random, and then every gene after this point is switched between the two chromosomes. This can be done also as a two-point crossover, in which two randomly selected positions divide the chromosomes into three sections. This list is by no means exhaustive, which underlines the variety of different methods. Similar lists could be generated for how to choose the chromosomes for reproduction, when to stop the algorithm, how to mutate the chromosomes et cetera. In short, different choices in implementing the genetic algorithm could lead to highly different solutions.

This became a problem when implementing the algorithm with the parameters presented in (Yu et al., 2007) produced quite weak results. For example, even with the test

cases presented in the article, the algorithm struggled to finish in the optimal results. This can of course be due to the implementation, as the source code used in the article was not available, despite trying to contact the authors. Trying out different approaches to implementing the genetic algorithm would have taken too much time in the context of this study.

In addition, the runtime of the algorithm at present and in the future was somewhat of an issue. As discussed, in the article the runtime for a DSM of 60 elements was "about a day". There are some factors that would affect this if applied at ABB. On the one hand, the article was written several years ago, which means some shortening of the runtime due to hardware development. On the other hand, in the short run the technology used to implement the algorithm would be VBA, due to the author of this thesis being unfamiliar with other technologies available. The efficiency of an Excel macro written in VBA would be nowhere near the efficiency of a compiled C++ program, which was used by Yu et al., and as such the runtime could be expected to be longer.

In addition, there was the question of changes in the DSM size. ABB's DSM is currently 54 elements, which is quite close to the DSM in the article. However, according to the BU IS manager, the DSM could be expected to grow to up to a hundred elements. This increase in the problem solution space affects genetic algorithms quite harshly; for example, in (Yu et al., 2007), chromosome pools of 3000 parents and 3000 offspring are used for a DSM of 22 elements. For the DSM of 60 elements the same numbers are 8500 parents and 8500 offspring. For each of these chromosomes, the objective function will have to be calculated, meaning increases in the runtime of calculating just one generation of chromosomes. Therefore, if the number of applications included in the DSM were to increase, the usability of the genetic algorithm would suffer.

The issues presented above are related to the search algorithm. However, as discussed in chapter 4.4.3, the objective function and the search algorithm are somewhat separate; thus, if desired, the MDL objective function presented by Yu et al. could possibly be combined with another type of search algorithm, for example the one used (Thebeau, 2001). This would of course require some work, but if the objective function was seen as useful, this avenue of research could be explored as well.

Yet, this idea was abandoned as well, due to observed deficiencies in the objective function. At this point, let us analyze the objective function formula presented by Yu et al. This is shown below:

$$f_{DSM}(M) = (1-\alpha-\beta)\left(n_c \log_2 n_n + \log_2 n_n \sum_{i=1}^{n_c} cl_i\right) + \alpha[|S_1|(2\log_2 n_n + 1)] + \beta[|S_1|(2\log_2 n_n + 1)]$$

The formula consists of three sections, which have been highlighted in different colors. The first part, highlighted in red, concerns the number of clusters in the solution. This part of the formula assumes that each cluster is fully filled with interconnections and no connections exist outside cluster boundaries; let us call the matrix constructed

based on these assumptions the ideal DSM. This situation is of course highly unlikely, which is why the two other parts, in blue and green respectively, represent the mismatches between connections in the idealized model of the DSM and in the real DSM. The use of 2-based logarithms represents the idea of minimizing the number of bits required to exhaustively depict the DSM.

The weights α and β represent the importance given to the three parts of the formula. Here is the first problem with this calculation, as the weights need to be given as parameters. In the original article the weights were set to imitate clustering arrangements created by experts for three DSMs. However, it is not clear if the same values would fit with the objectives of this study's case.

More importantly, a deficiency was observed in the function: although it does account for setting elements to multiple clusters and treating bus elements separately, it does not take a stance on the positioning of the elements within the clusters. This severely limits the usefulness of the multicluster allocation. The problem is elaborated below.

In Figure 7-1 is pictured one of the test cases used by Yu et al., a DSM with three overlapping clusters (Yu et al., 2007, p. 101). The leftmost matrix is the one given as input to the algorithm. On the upper row we have the ideal and actual DSM for the solution presented in the article. This solution has the lowest overall description length possible for this DSM.

However, on the lower row we have the same matrices, but element 7 has been

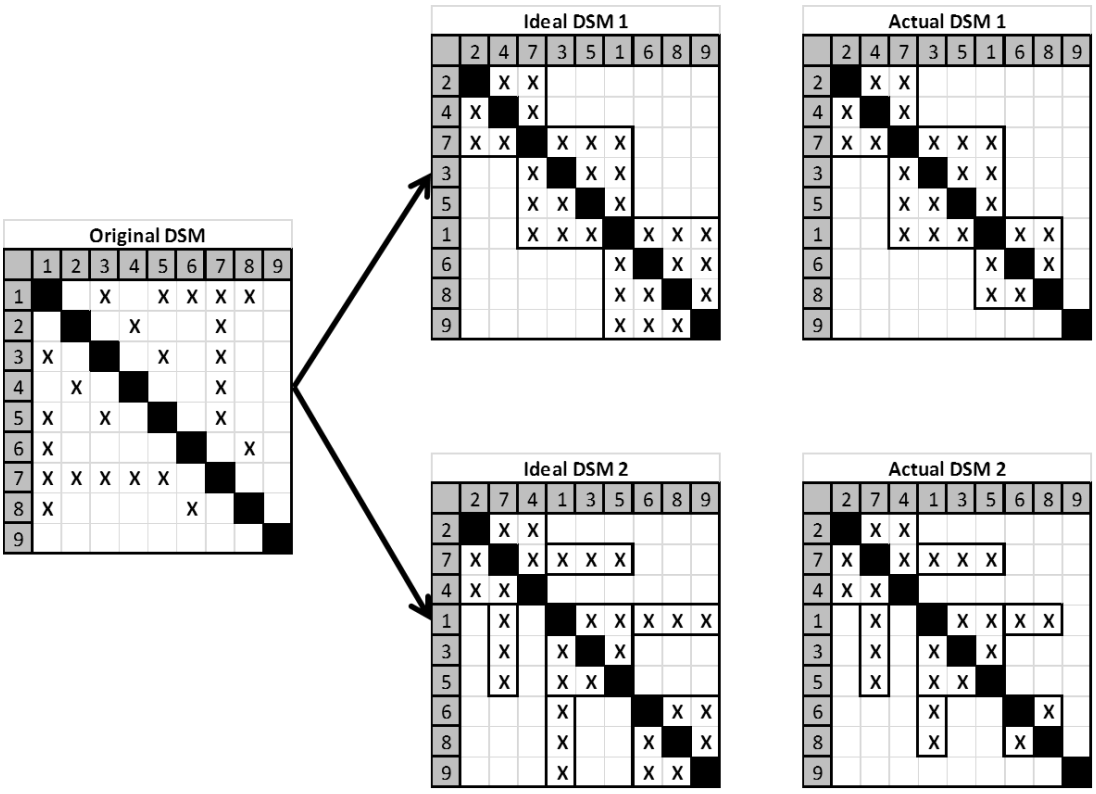


Figure 7-1: Ideal and actual DSMs found by the algoithm in (Yu et al., 2007) that have the same MDL value

placed before element 4 and element 1 has been placed before element 3. The elements that have switched places are the ones that have been placed into two clusters simultaneously. It is easy to see that the DSMs on the lower row are more difficult to understand. Yet, the MDL metric gives them the same values: the number of clusters is the same and they are composed of the same elements. Also there is the same number of mismatches between the ideal and actual DSM so the mismatch description length is the same.

In an idealized case such as this it would be easy for a person to rearrange the elements to find the best order within this clustering scheme. However, in a larger DSM it is easy to see that searching for the best order of the elements quickly becomes an optimization problem of its own, especially if an element is part of more than two clusters and the bus is introduced, as at this point it might be necessary to move the clusters as well.

As for how to solve this problem, (Yu et al., 2007) gives no indication. Of course, it is possible that the authors have solved the problem in their implementation of the algorithm, but as their code was not available, this could not be checked. As it is, if the algorithm were to be used, assigning elements to multiple clusters would be more likely to make the DSM harder to understand than if an element could belong to just one cluster.

Thus, at this point it was decided that the benefits of the approach of Yu et al. were outweighed by the disadvantages: the complexity of implementing the algorithm, the running time issues and the dubious results produced. Therefore efforts to implement this clustering approach were stopped, and the focus was set on the IGTA algorithm based on (Thebeau, 2001). This will be discussed next.

7.2 Implementing the IGTA algorithm

As discussed previously, the IGTA algorithm is a much-used stochastic algorithm for clustering a DSM. There is also a MATLAB implementation of the algorithm used in (Thebeau, 2001) publicly available (DSMWeb, n.d.). This was a great asset, which allowed using MATLAB instead of VBA for the implementation.

The algorithm aims to minimize the following cost function (Börjesson and Hölttä-Otto, 2012):

$$TotalCost = IntraClusterCost + ExtraClusterCost$$

Where:

$$IntraClusterCost = \big(DSM(j,k) + DSM(k,j)\big) * (ClusterSize_y)^{powcc}$$
$$ExtraClusterCost = \big(DSM(j,k) + DSM(k,j)\big) * DSMSize^{powcc}$$

Definitions for the variables in the formula can be found in appendix 3. In general, the algorithm balances between the number and strength of interconnections outside of

clusters, and the sizes of the clusters. This is different from e.g. Zakarian's algorithm that only attempts to minimize the number of connections outside clusters. The main output of the algorithm is a cluster matrix with the clusters in the rows and elements in the columns, which results in the lowest total cost found during the particular run of the algorithm.

The working principle of the original IGTA algorithm is as follows. Initially each element is set into its own cluster. After this, during each round of the algorithm, an element is chosen randomly. Then, each cluster makes a bid for the element (henceforth the bid element) to join the bidding cluster. In the algorithm used in (Thebeau, 2001), the bid is calculated based on the connections the bid element has with elements in the cluster making the bid. The bid element is then usually moved to the cluster with the highest bid; based on a probability given as a parameter, the algorithm randomly chooses the second best bid in order to avoid getting stuck at local optima.

It is only after this the total cost for the new clustering arrangement is calculated, in order to check if moving the element results in a lower-cost solution. Usually, if the proposed clustering arrangement results in a higher total cost, the change is discarded and another element is set as the bid element. Again, based on a parameter probability a worse solution is randomly accepted, but the previous solution is recorded to allow returning to it if the end result of the algorithm would be worse. This loop of accepting bids for elements and trying the new solutions for a lower total cost is continued until the termination condition is met; in the original algorithm it is simply a number of rounds with no improvement in the objective function.

As said, the original MATLAB implementation is available publicly. However, modifications to the algorithm have been presented in the literature. There will be discussed next.

Firstly, (Börjesson and Hölttä-Otto, 2012) present two modifications to the algorithm that, according to their data, improve both the efficiency and best total cost found each round. The first modification is the "improved termination condition" (ITC), which ensures that during each round, each element is bid for only once. This is an improvement over the strict random selection in the original algorithm, in which the same element could end up as the bid element several times during the same round, even though the total cost would not improve by moving the element. According to the data in the article, this modification improves the efficiency of the algorithm, while not taking away any functionalities, so implementing ITC was an easy choice.

The second modification is "suppressing multicluster allocation" (SMA), which enables using a more efficient calculation for the total cost by forbidding elements from belonging to more than one cluster. This modification required more thought, as it takes some functionality away from the original algorithm. However, if multicluster allocation were allowed, the same problem as with the algorithm of Yu et al. would arise: how to place elements within clusters? As discussed, in Thebeau's approach the elements are replicated for each cluster they belong to. This was considered to unnecessarily complicate the DSM. It could also be asked if multicluster allocation was even in line with the

objectives of this study; the aim is to find separable groups of elements, in order to reduce cross-module interactions. If an element was assigned to several clusters, it would immediately create dependencies between the clusters. In addition, (Börjesson and Hölttä-Otto, 2012) present data based on which the solutions found with the SMA modification applied are better than if multicluster allocation were allowed. Therefore, it was decided to implement also the SMA modification.

The second modification regarded calculating the bids for the elements. As discussed, in the original implementation, the bids are calculated based on the connections between the bid element and elements in the bidding cluster, and only then is the change in the total cost calculated. Intuitively it would feel more sensible to calculate the bid based on the change in total cost that moving the element would cause; this approach has been used in (MacCormack et al., 2006), who calculate the bid as the marginal improvement to the total cost.

However, no data could be found to back either claim. Therefore, to decide which approach to implement, both ways of calculating the bids were tested to compare the results. Initial tests with a small number of runs suggested differences in the total cost, but also in the number of clusters in the proposed solution; more specifically, the solutions found using the original method for calculating the bids seemed to propose a larger number of smaller clusters. This was decided to be included as a variable, in addition to the total cost. Intuition would suggest that a smaller number of larger clusters would be easier to comprehend for the user, and more useful as a basis for dividing the AA into modules. Thus, if there would be no significant difference in the objective function values presented, the suggested number of clusters could be used to make the decision between the bid calculation methods.

In addition, the running time of the algorithm was also included. This was an important factor with regard to the usability of the tool. If there would be no significant difference in the other variables, the running time could be used as basis for the solution.

For more rigorous testing of the variables discussed above, the clustering algorithm was run 10000 times which each method for calculating the bids. Due to time constraints, the data given to the algorithm was an earlier version of the DSM than otherwise in this study, with 58 elements. However, this does not pose a threat to the validity of the results, as the data are the same in all tests of the algorithms. The results are presented below, in tables 7-1 and 7-2.

Table 7-1: Comparison of bidding methods used in (MacCormack et al., 2006) and (Börjesson and Hölttä-Otto, 2012)

MacCormack et al. 2006			
Property	*Total cost*	*Number of clusters*	*Time to run alg. once*
Mean	3437	11	0,66
Median	3415	11	0,65
St. Dev.	92	1	0,07
Min	3229		
Max	4152		

Börjesson & Hölttä-Otto 2012			
Property	*Total cost*	*Number of clusters*	*Time to run alg. once*
Mean	3440	11	0,18
Median	3415	11	0,18
St. Dev.	94	1	0,02
Min	3201		
Max	4085		

Table 7-2: t-test results for difference of means between the results for the bidding methods

t-test for the difference of means			
Property	*Total cost*	*Number of clus-ters*	*Time to run alg. once*
Actual difference	3	0	0,48
Hypothesized difference	0	0	0
t-test p-value (df. 9999)	0,009	0,218	[Too small value for precision]

The standard deviation was calculated as sample standard deviation; it could of course be asked, if this dataset represents a sample or the whole population of solutions as the data are created just for the purposes of this analysis. Thus, if these were to be considered a sample, then the population would be infinite. However, as none of the results represent an exhaustive search of the problem space, it was decided that they would be viewed as a sample of solutions offered by the algorithm. In addition, for such a large amount of data, the values for population and sample standard deviation would be nearly the same anyway.

As can be seen in Table 7-2, the differences between mean values of the variables are statistically significant for total cost at the 0.01 confidence level and at an even greater confidence level for the difference between the running times; the exact value could not be determined, as the software used did not have high enough precision. No difference of statistical significance was found for the mean of the number of clusters.

However, one should ask, if the differences have any effect on the results; the t-test provides only information on whether the means are likely to be the same or not. The difference in the running time has a visible effect: for example, running the algorithm 10000 times using the bidding method from (MacCormack et al., 2006) takes about 110 minutes, or nearly two hours. On the other hand, using the original bidding method, the 10000 runs take about 30 minutes. Both are of course quite short times with regard to clustering algorithms, which sometimes run for several days. However, with regard to the usability of the tool, a shorter time is definitely better, if the same results can be achieved.

This brings us to the question of whether the difference of 3 points in the mean total cost values is something to consider. To this end, the spread of the total cost results data was analyzed to find out what was the effect of extreme values in the total cost mean; as seen already in Table 7-2, the median was the same for both datasets.

The method by (MacCormack et al., 2006) found 225 unique results, while the one by (Börjesson and Hölttä-Otto, 2012) found 247 unique results. However, the analysis provided information that the results created by both bidding methods are for the most part similar. Table 7-3 aggregates this information, showing that the same nine results account for 80.3 % of results in the first dataset and 77.2 % in the second. Moreover, the three most common results account for approximately 50 % of results in both datasets, and the two most common for 40 %. Thus, it seems that both bidding methods create a quite similar spread of results.

Table 7-3: The nine results representing approx. 80 % of results in both data sets

MacCormack et al. 2006		
Total cost result	**Count**	**Running total %**
3415	2364	23,6 %
3546	1835	42,0 %
3358	835	50,3 %
3326	731	57,7 %
3457	691	64,6 %
3485	684	71,4 %
3377	527	76,7 %
3320	205	78,7 %
3288	161	80,3 %
Grand Total	8033	80,3 %

Börjesson & Hölttä-Otto 2012		
Total cost result	**Count**	**Running total %**
3415	1971	19,7 %
3546	1935	39,1 %
3358	808	47,1 %
3485	778	54,9 %
3326	670	61,6 %
3457	566	67,3 %
3377	555	72,8 %
3320	231	75,1 %
3288	206	77,2 %
Grand Total	7720	77,2 %

As for the lowest solutions, the method by (MacCormack et al., 2006) found its lowest solution of 3229 nine times. (Börjesson and Hölttä-Otto, 2012) found its lowest solution of 3201 only once; however, it also found a solution with a total cost of 3229 eleven times. Based on this evidence, both bidding methods should be equally capable of finding good solutions. That information is enough to tip the choice between the

methods to the favor of (Börjesson and Hölttä-Otto, 2012), due to the much shorter running time.

The next question regarding automatic clustering was about the number of runs needed. Heuristic algorithms such as the one used here are typically run a number of times to ensure that a good solution has been found. The difficulty is in the randomness of the results; because all solutions are not tested, the algorithm could find the global optimum solution on the first run, or it might not find it at all, regardless of the number of runs. The question of how to optimize the number of runs is a matter of active research, but no conclusive answers exist as of yet. (Kanniainen, 2013)

The original algorithm presented in (Thebeau, 2001) utilizes two random factors in order to avoid local optima. However, (Börjesson and Hölttä-Otto, 2012) claim that the clustering algorithm will tend to get stuck despite the measures, and as such the algorithm will have to be run a large number of times in order to be sure that a good solution has been reached. They cite a number of 10 000 runs as enough to produce reliable results, yet they give no arguments to back this number.

Increasing the number of runs naturally causes running the algorithm take a longer time in total. This is not as big a problem as it used to be, as running the algorithm in MATLAB 10000 times takes only about half an hour for a DSM of roughly 60 elements, depending of course on the hardware MATLAB is running on. Still, if reliable results can be achieved in shorter time, it is always for the better. Another question is if the user can be sure if even a large number of runs is enough. Therefore, for practical purposes, the future user of the tool was interested in some evidence on a suitable number of runs, a "rule of thumb" to guide the use of the automatic clustering.

As discussed previously, on 10000 runs the algorithm only managed to find the solution of 3201 once. However, there is no certainty that the next 10000-run set would find as good a solution or that this solution is the global optimum. An approach would be to increase the number of runs even further, say, to 100000, to see if there is an even better solution. Yet, increasing the number of runs ever further does not seem like a very fruitful method of testing the algorithm.

As presented in Table 7-3, a quite small number of results forms the main mass of results found by the algorithm. A possible method of analyzing the number of runs needed was to compare the sensitivity of the DSM clustering scheme to the total cost found, i.e., to move the object of analysis from the total cost to the actual DSM. If there would be little difference between the best result found and for example the median result, then a number of runs smaller than 10000 could be enough to produce satisfactory results.

The original MATLAB implementation of the algorithm offers a tool for this purpose. The function compares each pair of cluster and run, and returns the similarity between the clusters. This similarity is called the likeness of the clusters by the original author. The calculation is done so that the number of common elements in both clusters is multiplied by two, to account for that each common element appears in both clusters, and then divided by the sum of the number of elements in each cluster. This provides a

percentage value for the similarity of the two clusters. These values can then been aggregated into average similarity of a cluster to the best matches across all runs, which can further be aggregated into the average similarity of a run to all other runs.

However, for these calculations, the number of runs analyzed would have to be much lower than the 10000 runs tested earlier. As each cluster of each run is tested against all other clusters and runs, the number of calculations to perform scales very strongly; for example, for 100 runs of 10 clusters on average, $1000 * 1000 = 10^6$ calculations have to be performed and recorded. Handling datasets of this magnitude on a normal workstation easily causes errors due to the system running out of memory. In the end, a 400-run set could be gathered for the purposes of this thesis.

Sadly, the clustering scheme for the result 3201 was not available, and thus the best result found in this set was 3229, with one hit. The worst solution had a total cost of 3952. Interestingly, the nine results presented in Table 7-3 again accounted for nearly 80 % of results, as seen in Table 7-4. This is additional evidence that at least for the source data used here the algorithm functions in a quite stable fashion.

Table 7-4: The nine most common results in the 400-run set

400 runs		
Total cost result	**Count**	**Running total %**
3546	83	20,8 %
3415	80	40,8 %
3358	29	48,0 %
3485	28	55,0 %
3326	26	61,5 %
3377	24	67,5 %
3457	24	73,5 %
3320	8	75,5 %
3288	8	77,5 %
Grand Total	310	77,5 %

To compare the effect that changes in the total cost have on the clustering scheme, the clustering solutions were compared according to the method by (Thebeau, 2001). Firstly, from the aggregated data on cluster similarities, it was found that on average the solutions found by the runs were a bit over 90 % similar; to achieve this result, firstly the best match fractions for each cluster in each run over all other clusters were averaged to find out the average likeness of each cluster with its best match in all other runs. These values were then averaged to find out how similar each run was to each other run. By averaging these values, we arrive at the average likeness of all runs in the dataset. The average 90 % likeness is already quite high, as it suggests that the spread of the total cost from 3229 to 3952 was caused by changing only 10 % of the clusters.

However, the average value can of course hide some information, so more specific calculations were performed for the best solution (3229), worst solution (3952) and the

two most numerous solutions found (3546 and 3415). As there was more than one of each comparison run, a representative was chosen at random from among these. The data revealed that all of the solutions with these total costs had also the same number of clusters in their solution. Even though this does not guarantee that the solutions are the same, they can be assumed to be similar in their composition, considering the 90 % average likeness of all runs.

The results for the similarity of the best solution with the other solutions are shown in Table 7-5 below. In addition, the corresponding DSMs are shown afterwards in Figure 7-2 for comparison.

Table 7-5: Similarity of best found solution with other choice solutions

	Similarity with (run #, result)		
Best result (run #, result)	*191, 3415*	*39, 3546*	*164, 2952*
55, 3229	95,2 %	91,7 %	84,7 %

Comparing the clustered DSMs, firstly, the difference between the best and worst solution is quite visible, as foretold by the lower similarity than with the other two solutions. The worst solution consists of almost double the number of clusters as the best solution. This naturally leads to smaller clusters, and a more scattered view to the architecture. The difference is not so striking with the two most common total cost solutions. Especially the solution of run 191, with a total cost of 3415, has several clusters that are immediately recognizable in the best solution. The difference in the total costs comes from the small differences in the clusters; for example, cluster 5 of the best solution and cluster 3 of the 3415-cost solution look similar, but the former is composed of 11 elements, while the latter contains only 9 elements.

Sadly, it is very hard to say anything more concrete than this on the required number of runs. If one of the two most common solutions is considered good enough, then a number of runs much smaller than 10000 is enough, as their shares of the solutions found seem to be constantly around 20 % for each. On the other hand, the first 10000-run set found the absolute lowest cost of 3201 only once. Therefore, if the aim is to find the absolutely lowest-cost solution, the only way to make the probability of its appearance higher is to increase the number of runs. At this point, it becomes a question of the amount of resources that are available for running the clustering. Luckily, the algorithm can be set to run for example overnight, if desired.

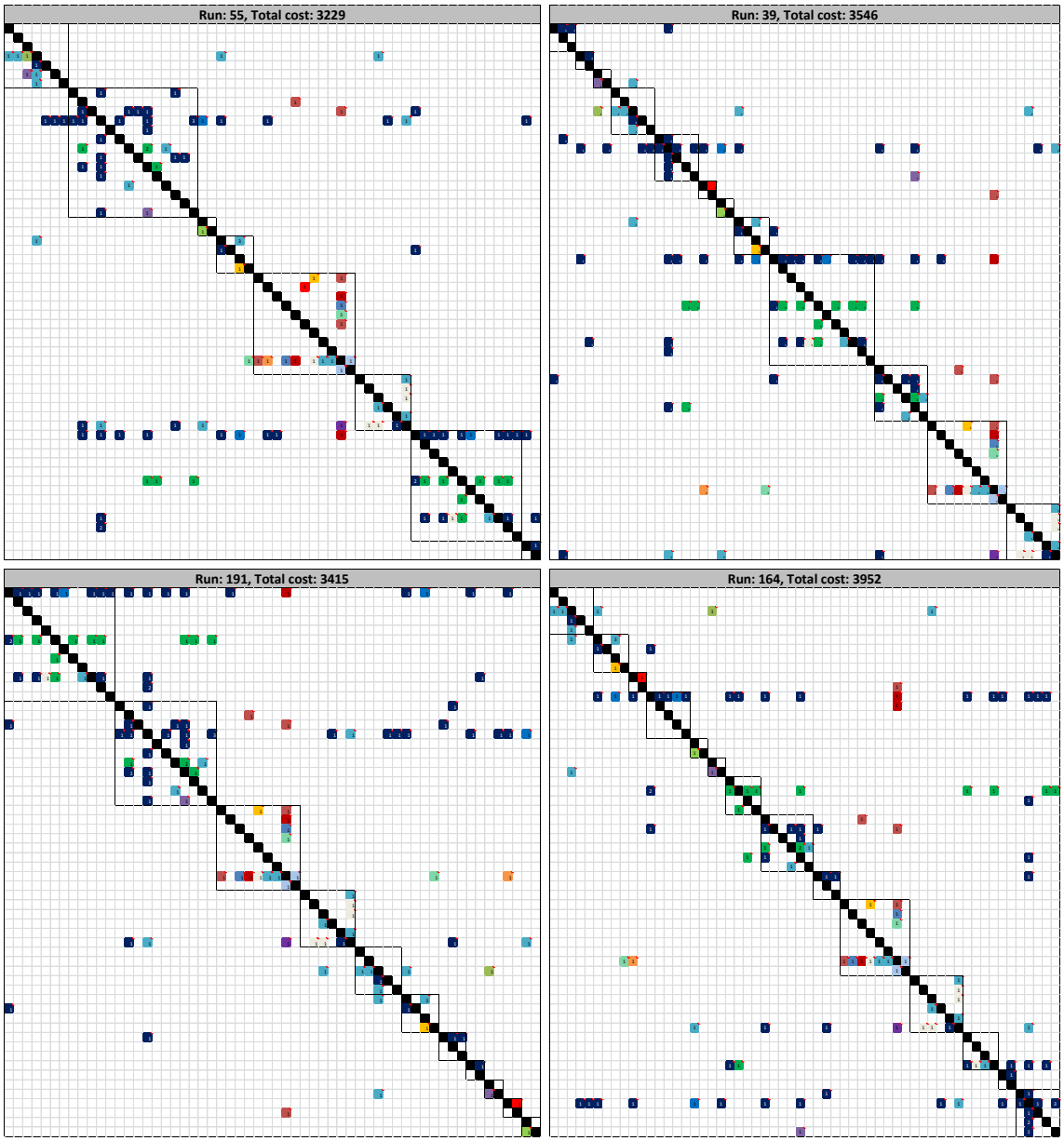


Figure 7-2: Comparison of clustering schemes with choice total costs

8 COMBINING AND MEASURING THE PROPOSALS

There are two questions set for the development in chapter 4 that have not been answered yet. Firstly, there is the question of how to combine the suggestions presented by sorting the DSM by attribute values and by performing clustering on the architecture. Secondly, a decision has to be made on what metrics to set on the architectural suggestions to measure the "goodness" of the architecture presently and as it develops. These will be discussed next.

8.1 Combining architectural suggestions

As discussed, the architectural suggestions created by sorting with attribute values and by clustering represent two different views to architecture modularity. The former, based on the attributes, reflects the aim of cohesion; that the applications included in a module belong logically together. The latter architectural suggestion created by the clustering algorithm represents the view of coupling, i.e., it is focused on the connections between the applications.

Both views can be seen as valuable in their own right. However, if some way of combining the information in the views could be found, it would further improve the reliability of the architectural suggestions, as then both views could be taken into account. However, this is easier said than done, as the information provided by the views is conceptually quite different.

For the purposes of this study, combining the views was implemented by using the results of each suggestion as input for the other. This is further elaborated next. Firstly, consider the business module – based view to the architecture. This represents the cohesion side of architecture modularity. An important problem with regard to dividing the applications to business modules was that several applications could be included in several modules almost equally well; this was due to an application e.g. serving several purposes or being composed of several modules, as with ABB's ERP system. While this problem could be solved to some degree by allocating an application to multiple modules, this was decided against due to it making the DSM more complex, as discussed in chapter 7.2. As such, the business module division represents one possible scheme of allocating the applications, and it can change in the future.

Therefore, to check if there would be a more logical division of the applications that still has an aspect of the cohesion between the applications, the clustering algorithm was implemented so that it can take as an input an initial clustering scheme. As discussed,

normally the algorithm initially places each application to its own cluster, and works from there. The algorithm only deletes clusters during its course, it does not add them. Therefore, if given e.g. the business module division as an initial clustering scheme, the algorithm could move elements between the clusters and delete clusters, but not add clusters. This ensures that the solution will consist of the same or a smaller number of clusters, and will therefore be of similar complexity with regard to the number of clusters as the initial scheme. The aim is an “improved version” of the business module allocation.

Similarly, the clustering algorithm accounts for bus elements, if they have been allocated. This is implemented so that if a bus is used, the DSM is clustered only up to the bus.

The effect that automatic clustering has on the original business modules division is shown in Figures 8-1 and 8-2 below, with the clustering based on the business modules data in the former and the result of the clustering algorithm with the data as input in the latter. The clustering algorithm was run 10000 to achieve this result. The DSM also con-

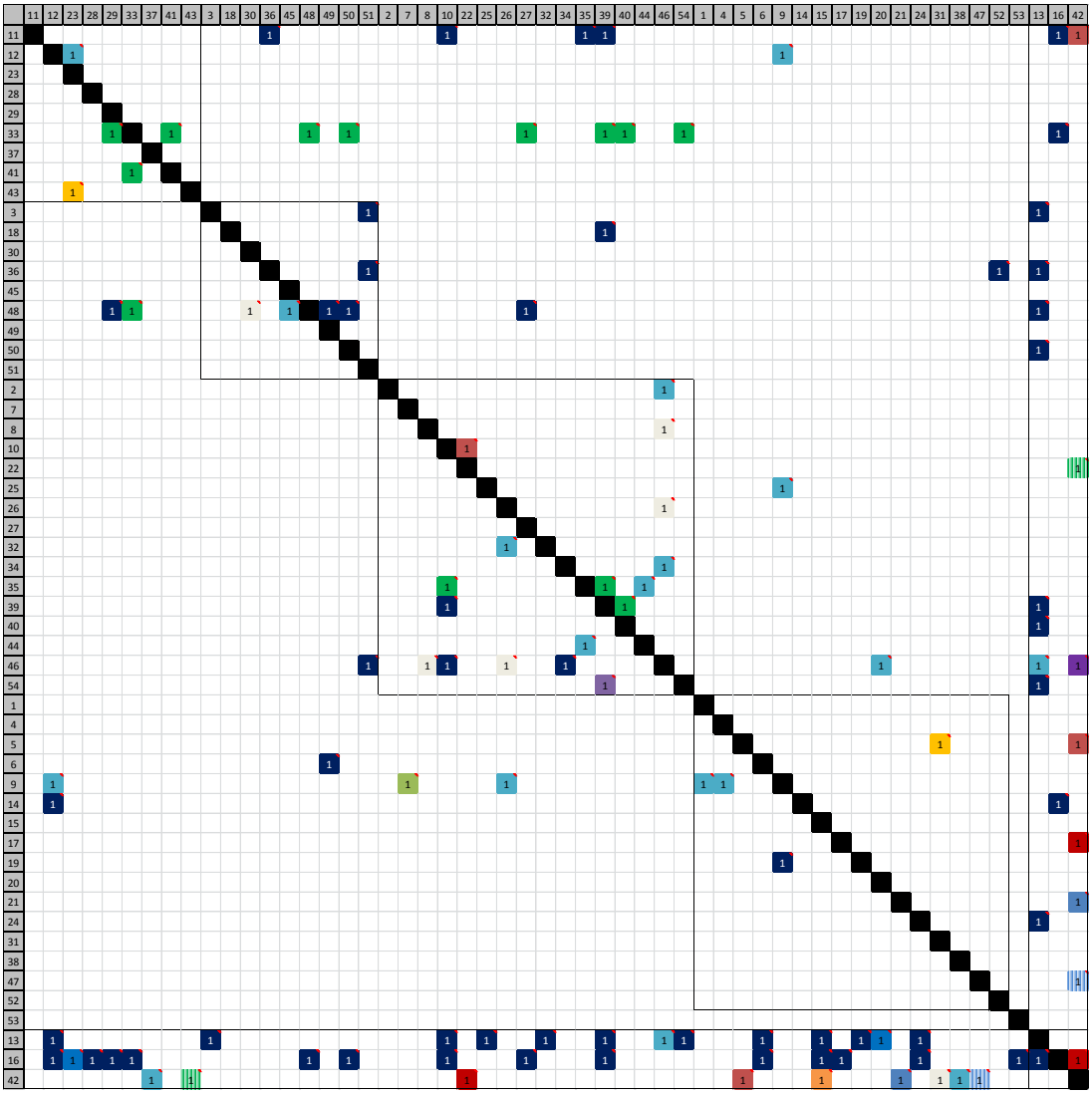


Figure 8-1: DSM clustered based on business values data

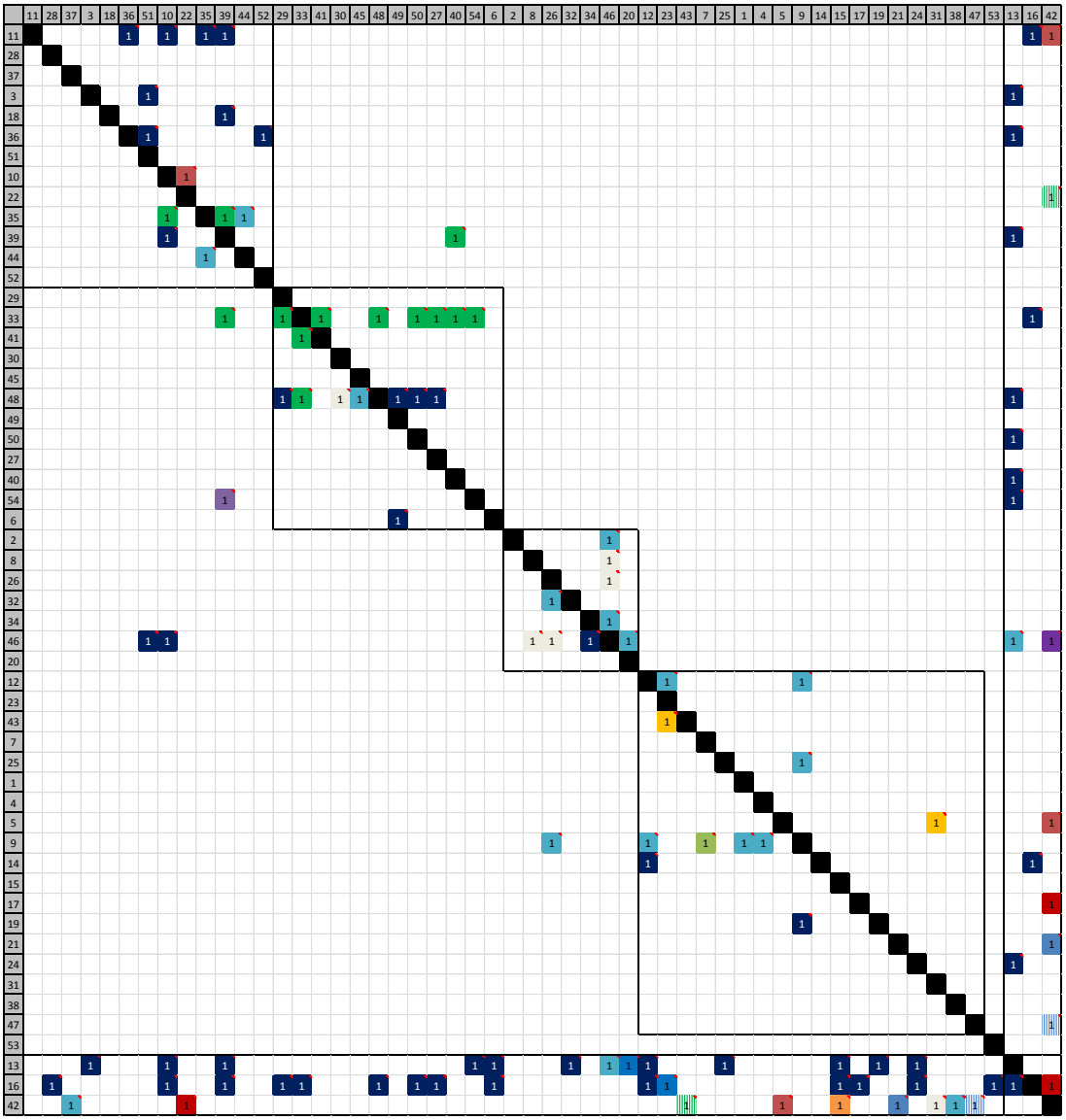


Figure 8-2: DSM created with the clustering algorithm with the business modules data as input

tains a bus for the sake of example.

As can be seen, there is quite a marked difference between the number of connections outside the clusters, the total sum of which in the former being 24 and in the latter only 6. Also the likeness between these clustering schemes, calculated as discussed previously, is only approximately 60 %, which is markedly lower than the likeness of runs produced previously. Thus, the clustering algorithm has made significant changes to the clusters based on the business modules. Combined with information on the applications themselves to ensure cohesion, this clustering scheme improved with regard to the connections, can now be used to develop the classification of the applications, if so desired.

The second way of combining the views is that the results of the clustering can be subjected to the interface search described in chapter 6.3. Therefore, another architectural suggestion can be generated. Whether this is based on the algorithm clustering the applications freely or using the original cluster division as base, is up to the user. With

this step, the result of the clustering algorithm can be used as the starting point for the development suggestion.

To elaborate this, a 10000-run set of data was produced with the clustering algorithm, with each element initially in its own cluster. The output clustering scheme was the imported back to the Excel tool and subjected to the interface search process. The result is shown below in Figure 8-3.

As can be seen, this particular clustering scheme did not produce a very sensible module division with regard to finding interfaces between clusters, as interfaces could only be found between modules 2, 3 and 4. Yet, it could still be used as a base for further modifications by moving applications between clusters, or as a starting point for development plans. From this point of view, setting some applications into the bus makes more sense if the clustering algorithm is used; as the algorithm aims to eliminate all connections between clusters, optimally there would be no connections that could be used as interfaces for the module-to-module connections. Allocating some applications

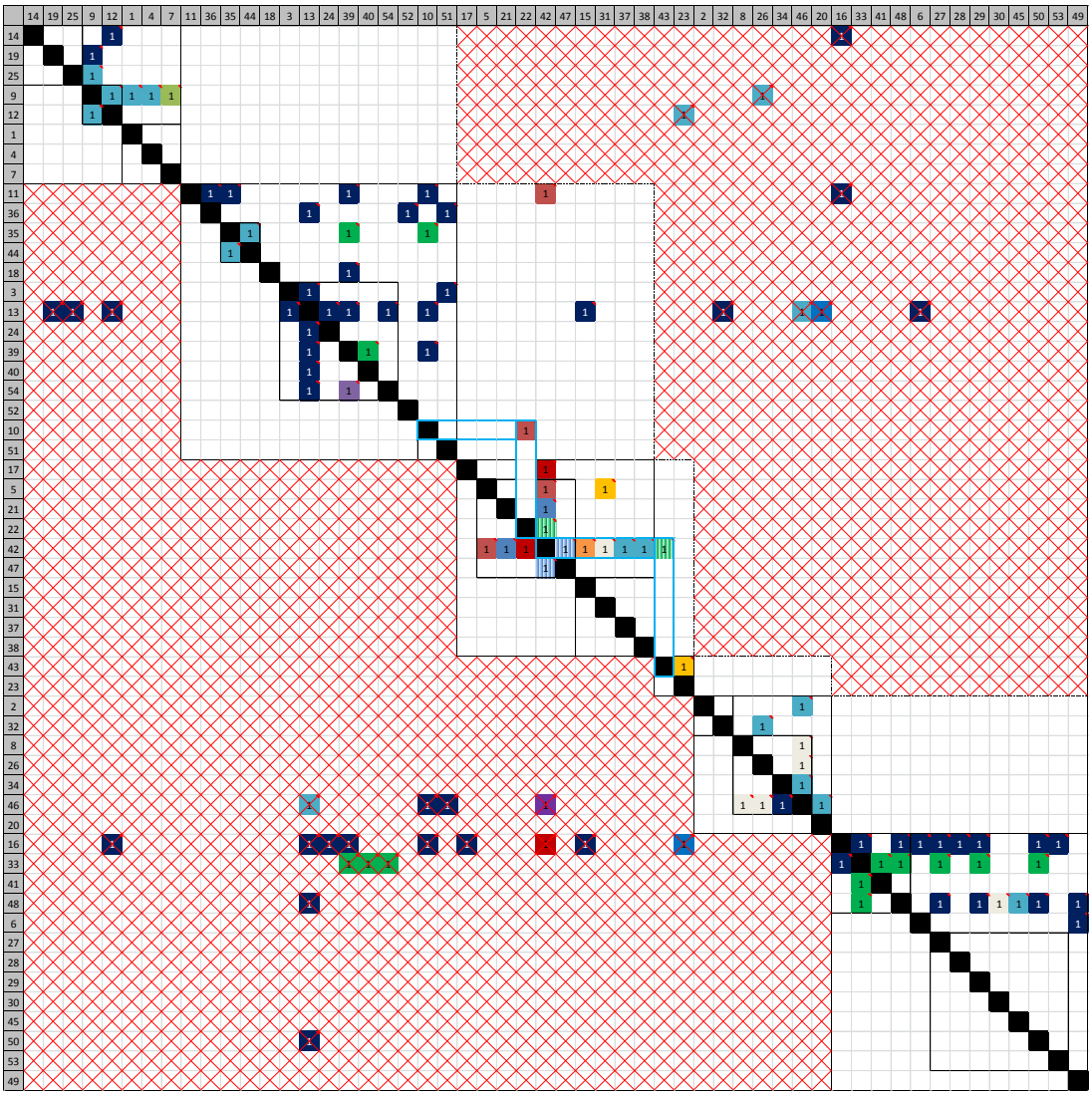


Figure 8-3: DSM produced by clustering algorithm used as base for searching interfaces

to the bus, so that interfaces could be formed through it, ensures that the aim of the clustering algorithm does not contradict the aim of searching for interfaces.

8.2 Measuring the suggestions

The last question regarding the implementation of the DSM tool was what to measure from architectural suggestions and sequential rounds of developing the architecture. As in all other areas of management, sensible metrics are important for setting goals and checking that progress is made towards the goal.

When this study was begun, there was interest in finding out if suitable metrics could be found in literature. If some metrics had been discussed previously, chances were that they could be in use in other organizations as well, and could therefore be used to benchmark results between ABB and other organizations. This way, information could be gathered on what constitutes a "good" enterprise architecture, and the development initiatives could be directed correctly.

Unfortunately, as discussed in chapter 2.4, despite numerous frameworks for guiding EA development, quantitative measurement of the architectural plans is something that is left for the organizations to figure out for themselves. The same goes for the subareas of EA work discussed in this thesis: application, data and technology architecture. One could ask if presenting universal suggestions for EA development is even possible; as the goals of the IS function varies considerably between organizations, so should the focus of measurement. For example, the EA development focus examined in this thesis concentrates on increasing the system's flexibility. However, this necessarily comes at some expense to the performance of interactions between programs, which could be the more important objective at some other organization. Even within the architectural goal of increasing flexibility, the variables depending on the context quickly become so numerous that universally applicable guidelines could be hard to present, let alone hard numbers to aim at.

As for what could be measured from the DSM, some initial ideas found in literature were discussed in chapter 4.4.2. Firstly, the size of the core has been deemed as important in several articles that discuss using the DSM both in analyzing a single program and EA. However, despite being interesting and having some theoretical background, there has been little discussion on how to actually make the core smaller, or other cyclic groups for that matter; this is a still lacking area of research identified e.g. in (Lagerström et al., 2013b, p. 16). Thus, even if the size of the core could be measured, no concrete development proposals could be made based on the measurements. This limited the usefulness of this metric with regard to the focus of this study.

Another type of metric which was discussed is the variety of metrics of system modularity. As part of implementing the clustering algorithm, the metric from (Yu et al., 2007) was tested and the one from (Thebeau, 2001) also implemented. Yet, in this case, the interest in using these metrics is limited to the clustering. The metrics can present some indication of how modular the current architecture is and what the effect

would be if an element were moved to a different cluster. However, with regard to the desired development suggestions, this information is not very useful.

With regard to the views created by the tool described in this thesis, some potential metrics stand out. First, an obvious one is the number and strength of connections outside clusters in the interface suggestion. For the forbidden connection zone, this metric is conceptually simple. However, also in the allowed connection areas, every connection beyond the first one is in contradiction of the aim of architecture development: that each pair of modules would only be connected through a single interface. Thus, an optimal value for this metric would be equal to the number of module pairs.

However, due to the flexibility in creating architectural suggestions, this metric is problematic; more specifically, the choice of bus applications strongly affects the number of connections in the allowed and forbidden connection areas. Thus, the person creating the suggestions has great influence over the number of connections simply by the choices of bus applications. On the other hand, over several rounds of developing the architecture and using the same bus applications, this metric can become meaningful, to track the effect of development projects.

Another metric that was considered interesting is the nature and variety of transmission technologies. This view separates the DSM to two parts. Firstly, at this point the clusters used as basis for finding interconnections can be thought of as black boxes; as the aim is to reduce the dependencies between clusters, the choices of transmission technologies within the clusters at one site should not affect the choices at other sites. In other words, different sites should not be aware of the transmission technologies used within clusters at other sites. However, the transmission technologies used between application clusters are important. A suitable metric for these would be for example the number of different transmission technologies used to transmit information between modules: each additional transmission technology used at one site makes it more difficult to harmonize the connections between sites. Also the technologies used within the bus are important, if a bus is defined, as in that case moving the information is simply achieved by using the bus. Another option to the number of different technologies would be to set complexity values for each transmission technology and measure the sum or average value of these; this approach is possible with the tool, as described in chapter 5-4.

9 CONCLUSIONS

This chapter firstly discusses what conclusions can be drawn based on the study from both a theoretical and practical viewpoint; these aspects are discussed in chapter 9.1. Afterwards, in chapter 9.2, the reliability and validity of this study are discussed, and the contributions are assessed against the initial goals. Finally, in chapter 9.3, future research paths that were identified during the course of this study are discussed.

9.1 Theoretical and managerial contributions

The theoretical gap that is behind this thesis is the typically abstract nature of EA work. Many frameworks offer guidelines on what information to collect, but leave the analysis of the data to the organization. Thus, the results often remain at a very abstract level, and do not have any effect on real life in the organization. The same goes for TOGAF, the framework in use at ABB. Practical guidelines are needed on how to use the data collected as basis for concrete development plans. This type of research has obvious implications for managerial reality as well.

The primary theoretical contribution of this thesis is combining various DSM-related approaches found in research literature and applying them in the context of enterprise and application architecture development. To date there have been few previous works discussing the use of DSMs in the EA frame, despite similar matrices being present in many architecture frameworks. This thesis presents possible approaches on what can be done with the data after collecting it.

The central products of the tool are the interface suggestions, built either around module-to-module communications or connecting the applications through an enterprise bus. The thinking behind these objectives should be applicable to most other organizations; as discussed, initiatives to divide the EIS into separate application domains have also been present in literature. The novelty of this thesis is in the way several methods of DSM analysis are used in combination for creating these interface suggestions.

The transmission technologies used were also incorporated to the DSM. This type of analysis was not present in the literature reviewed for this thesis. Additionally, the importance of the transmission technologies can be hypothesized to be an important differentiating factor between analyzing a single program and an EIS. Due to the higher complexity of connections in an EIS, each connection is not the same; the technology used could have a significant impact on the strength of the connection. This represents an interesting new research path.

This thesis also presents some new information on popular DSM clustering algorithms. At least for the author the noticed deficiencies in the algorithm by (Yu et al.,

2007) were interesting, after the initial frustration. In addition, further comparative data are presented on alternative ways of implementing the IGTA clustering algorithm, and the number of runs that should be performed.

Regarding managerial implications, the views discussed represent concrete and visual plans for developing the architecture. Hopefully, with the methods presented here, the data that are collected during the EA work can be put to better use. In addition, the tool that has been developed as part of this study offers a user-friendly way of testing the different scenarios for future architectures.

9.2 Assessment and limitations of the study

9.2.1 Reliability and validity

The two main questions to assess the quality of a study are the reliability and validity of the results. The former means that, using similar data collection methods, other researchers would arrive at similar conclusions. The latter means firstly that data collection methods measure what they were intended to, secondly that the findings truly reflect what they claim to. (Saunders et al., 2009, pp. 600, 603)

As for the reliability of the study, there are two aspects to consider. There are concerns of the quality of data on ABB's information systems, as discussed in chapter 5.1. Firstly, during the study, deficiencies and errors were discovered in both the application and the interface portfolio, due to for example erroneous understanding of the data collection form. Secondly, the data added during this study on the business modules and levels of application ownership represent the views of a very limited group; another method of collecting these data could have led to a different allocation of applications to the modules. In addition, with regard to the DSM, the analysis on the optimal number of runs cannot be considered very rigorous, due to limited analysis of the similarity of the clustering schemes; the results are presented more as guidelines to the future users of the tool.

Yet, as also discussed, a factor mitigating these concerns is that the data in question are not the study's primary focus; the main result is the conceptual model, and no conclusions are presented that would be based on case-specific data on the information systems. In addition, the methods of creating the architecture proposals have been documented and are highly repeatable. The quality of these data will naturally have to be addressed or taken into account when the architecture proposals are used, but in the context of this thesis, their reliability is not a serious concern.

Another aspect of reliability is whether the functionalities included in the tool represent the requirements of the future users. As discussed, the development of the tool and of the method for creating the proposals was undertaken in close cooperation with the person responsible for this study at ABB. The communication between the author and this person was not very structured; rather, ideas were exchanged as part of normal workday discussions. This felt natural, as there were only two persons involved. As a

downside, this means that little documentation exists on the discussions, and thus replicating the same path of development would be difficult. A more structured way of doing research would have been to record the conversations on tape or in writing, in keeping with good practices of qualitative research (Mays and Pope, 1995).

On the other hand, most of the discussions concerned practical details of the tool, and as such might not be very interesting to include in this thesis. The arguments for the critical decisions, such as the foundations for the architecture proposals, have been presented in the text for analysis. Thus, a reader should be able follow the reasoning behind the proposed method, even though the exact way of arriving at this method is not discussed.

A factor affecting both aspects of reliability is that the data used in this study is based only on a single case. While this ensures that the solutions are close to the needs of the case company, it limits the generality of the findings. On the other hand, again, the results presented here describe a method of performing analysis, not the results of this analysis. Thus, the same methods can possibly be implemented in other organizations.

Based on the above, it can be summed that there are questions concerning reliability, but their impact on this study is limited. However, the focus on creating a method for analysis highlights the importance of validity. If the proposed method is not valid, then this study does not have much to give.

Identified questions on this study's validity have been listed below. Each area has a summary headline in bold and a more comprehensive explanation in plaintext.

Focus on analysis method, not value

The focus on analysis methods leaves the question of whether the proposed approaches will have actual value, for architecture development or ABB's business. This will only be seen once the methods have been used at ABB for some time. Yet, it was known from the start that this study only represents the first step in a more structured approach to architecture development. As such, further development is to be expected.

Only a part of EA involved

As for analyzing EA, this thesis concentrates on only a small part of architectural development. The data cover only a subset of applications in use at ABB, and considers only one business process. Even more importantly, the business architecture is ruled outside the scope of this study. This limitation was made even before beginning this study, as it was the opinion of ABB that the business architecture could not be changed during the course of this study and should therefore be taken as a given.

One could argue that this goes against the very idea of EA, as again the applications live their own life apart from the business requirements. However, there are counterarguments to this. Firstly, even the EA frameworks often emphasize that they should be used only in a scope that is in suitable for the resources available. There is experience also at ABB that grandiose plans of an all-encompassing redesign of the organization's

EA never lead to anything else than that: plans. This is why the scope was set to a level that was considered realistic. Secondly, the goals of this study are very much in line with the goals of ABB's business management. Thus, even though the business architecture is not explicitly present in the analysis, it was the foundation for the development proposals created by the tool.

Deficiencies in the practical implementation

Concerning the practical implementation, it can firstly be questioned whether the clustering algorithm chosen for the tool was the best alternative. To lessen this risk, the initial selection of tools was based on a reliable reference. Afterwards, the methods were compared to ABB's needs and also tested. The one implemented by (Thebeau, 2001) has been criticized in literature (e.g. Sharman and Yassine, 2004), and it has its shortcomings in producing clustering schemes that would reflect reality. However, the choice was only made after deficiencies in the more complex algorithm by (Yu et al., 2007) were discovered.

Secondly, the method used in sequencing at the moment can identify cyclic groups, but not if these contain smaller cyclic groups. The way this issue affects validity is that information on the cyclic groups is left out; this could affect the interpretation of the architecture proposals. There are several other methods for sequencing that do not have this limitation; these can be examined if further development is to be performed.

References more on practical methods, less on goals

A weakness in the theory used is that it is weighted quite strongly towards the technical aspects of DSM analysis, whereas the goals and development criteria for the architecture are based on ABB's areas of interest. Thus, the theory used does not have a very strong stand on what should be done with the architecture.

On the one hand, using ABB's goals for architecture development as the basis should link the suggestions produced to real needs. On the other hand, it would also have been preferable to find theory on how EA should be developed, or what makes for a good architecture. This would have given backing on the development proposals produced by the tool, and perhaps made possible comparing the current architecture with a benchmark. Yet, the theory on EA or EIS flexibility that was found often had a much wider focus than this thesis, or the results would have been difficult to modify for use with the DSM. In addition, literature using the DSM mostly studied individual programs, and the results were therefore not entirely useful for the purposes of this study.

Theoretical nature of proposals created by the tool

Lastly, an important general limitation is that the development proposals created by the tool are strongly theoretical in nature: it is highly unlikely that any interface proposed by the tool could be used in the intended purpose without modification. On the one hand, this is in line with what ABB wanted from the tool; the goal was to create theoretical development proposals that would then be modified into the practical plans. On the

other hand, this sets a great deal of responsibility on the person using the tool; therefore, it should be understood what DSM as a method can and cannot do and what have been the design decisions guiding the creation of the tool presented. This might not be a problem, as long as the person using the tool at ABB is the one who has also been involved in this thesis; yet, if this person changes or if the tool is for example taken into use at a different site, then there is a risk of incorrectly interpreting the recommendations given by the tool. If the user has any experience with the system being analyzed, there is probably a greater risk of the tool being abandoned rather than incorrect development decisions being made.

9.2.2 Research questions

To assess whether the study reached the goals set for it, let us consider the research questions set in chapter 1. Another thing to keep in mind is the aim for the architecture development, as discussed in chapter 4.1: the goal was to make the architecture more flexible, as defined also in chapter 4.1. This affected the areas of analysis present in the tool.

The first research question was what assumptions and limitations should be kept in mind when using the DSM. This viewpoint was discussed in chapter 3.2.3. The DSM is quite flexible as a method in the sense that its usability and limitations are strongly dependent on what source data are used, and at what granularity the results are shown. This became apparent during the course of this study. For example, at one point it was felt that depicting databases in addition to the applications in the DSM would present a more understandable picture of the architecture. However, already during data collection, it had been decided that the databases would be included in the connections; as discussed, in ABB's DSM, a single connection may span several steps of moving data. This makes for a different view to the architecture, and was only realized after discussing with an employee that was part of collecting the original data.

In addition, there are situations where other methods of analysis or visualization would perform better, such as in the case of three-dimensional architectures that were discussed. For the purposes of this study, however, the matrix representation was considered as satisfactory.

The second question was on what source data were needed to build the tentative to-be plans for the architecture. This was strongly affected by the aims that ABB had for the tool. Thus, as the initial interest was in dividing the applications by phases of the order-delivery process, these data were added to the database. In addition, information on the business levels was added. In the end these data were not used as intended, as discussed in chapter 6.3. However, they can still be used as part of creating the practical development plans, and possibly as part of future development. Lastly, the transmission technologies were also brought into the DSM, as they were recognized as an area of interest. The tool has also been built so that other types of connection attributes can be input into the DSM for examination.

Thirdly, the development criteria were selected to allow the user flexibility in creating different scenarios for future architectures, while being grounded in the literature on EA and EIS. The first criterion is that the future architecture should be fully hierarchical; the arguments for this were discussed in chapter 6.3, and it is present in the way the DSM is divided into allowed and forbidden connection zones in the interface suggestion. The other criteria can be affected by the user. Firstly, the division of applications into clusters can be done based on any attribute set for the applications, or it can be based on the results of the clustering algorithm. Secondly, the development suggestion can be built around connections from module to module, or on top of an enterprise bus. These criteria were chosen based on both the examined literature and ABB's views.

The last research question was what metrics should be used for the criteria. Sadly, this area did not go as far as was planned originally. This was discussed previously in chapter 8.2. The main problem was the lack of theory on suitable metrics to guide architecture development. Perhaps through the future use of the tool more metrics that are useful for ABB will be recognized. On the other hand, the dearth of literature could possibly have been eased by applying theory from software development in general, as architectural analysis has a longer history in that field. However, due to time issues and the field of software architecture being unfamiliar to the author, this research path could not be followed.

Thus, it can be said that the research questions have been answered, or a conscious decision has been made on the limitations in the answering. As for the research objective, what ABB looked for was a "step-by-step guide for using DSM in developing application architecture, taking into account the possible limitations of DSM and offering relevant development criteria based on literature and the needs of the company". The end result was not a step-by-step guide, but a concrete and implemented tool, which will hopefully facilitate the analysis in the future. The other sections of the objective have been discussed related to the research questions above.

9.3 Future research

Finally, this chapter discusses possible future research paths that were recognized during the course of the study. Firstly, more research is needed on whether and how EA actually leads to business benefits; this was discussed in e.g. Tamm et al., 2011. The same goes for the proposed development suggestions presented in this thesis. The methodology proposed here is just a first step; more research with data from a longer period of time is needed to verify if the methods actually have practical value, with regard to architectural development or ABB's business.

Secondly, it would be interesting to study how useful the results of this thesis would be, if the number of analyzed applications were greater. How would it affect for example the runtime of the clustering, and, perhaps more importantly, would the information needs for managing the architecture be different? With the quite limited number of applications that were analyzed in this thesis, it is possible for the architect to be familiar

with each of them. However, if there were e.g. 500 applications under study instead of 54, would different data be needed on the applications and interfaces, and should the data be analyzed differently?

A much larger number of applications would also further complicate the data collection, considering the manual method in use at the moment. These complications, along with the risks of errors in the data, could be mitigated if the collection could be automated; this is another area for future research. Methods for automatic EA data collection are currently under research, and some authors have already presented results; one example is in (Waldman and Sangal, 2007). The complexity comes from combining information from several technologies used in implementing the applications and databases that make up the technical portion of the enterprise architecture.

Tracking the system dependencies automatically would allow more rapid iteration of the architecture, and analysis of smaller units than entire applications. This way, the information could be used to manage the application architecture in a way that is more comprehensive and closer to the operative reality of software development at ABB. A theoretically interesting subject would be how this would affect for example the number defects in software, and how this information could then be used to make software support at ABB more efficient

Thirdly, the visibility matrix and the cyclic groups discussed previously seem like important factors with regard to the architecture's ability to accommodate changes. However, the way visibility is used in literature is still in the descriptive phase; what are needed are concrete suggestions on how to assess the risk the cyclic groups pose, and what should be done remove elements from these cyclic groups. During this study, there was interest in examining the effect of different measures of centrality on the cyclic groups; for example, bridge centrality measures how often an element is on the shortest path between two elements, which, intuitively, seems like it could have an effect on the cyclic groups.

A fourth area of interest is the automatic clustering. Taking other matters discussed in this thesis as a given, a more comprehensive test of different algorithms could be performed to see which produces the most useful results. In addition, the number of runs and its effect on the results of the used algorithm could be investigated more rigorously. An interesting question is e.g. the tradeoff between the quality of the results and the speed of generating alternative clustering schemes.

As a fifth avenue of future research, the significance of the transmission technologies with regard to the management of the information systems could be studied. In the context of ABB, these were considered as being of high importance. However, future research could focus on the effect of certain technologies on for example the maintenance needed on the connection. This would translate quite easily to which technologies are easier and cheaper to manage than others.

Lastly, as discussed, the analysis could be stretched to account for a larger portion of EA, for example, by incorporating the business architecture. This would provide a more comprehensive view to the architecture, as per the aims of EA initiatives in general.

REFERENCES

ABB, 2012. The ABB Group Annual Report 2012 156.

AIS Senior Scholars Consortium, 2011. Senior Scholars' Basket of Journals [WWW Document]. URL http://home.aisnet.org/displaycommon.cfm?an=1&subarticlenbr=346 (accessed 8.21.13).

Avritzer, A., Paulish, D., Cai, Y., Sethi, K., 2010. Coordination implications of software architecture in a global software development project. J. Syst. Softw. 83, 1881–1895.

Baldwin, C., Clark, K., 2000. Design Rules: The Power of Modularity, Vol. 1. The MIT Press.

Baldwin, C., MacCormack, A., Rusnak, J., 2013. Hidden Structure : Using Network Methods to Map System Architecture.

Banker, R., Slaughter, S., 2000. The moderating effects of structure on volatility and complexity in software enhancement. Inf. Syst. Res. 11, 219–240.

Bidan, M., Rowe, F., Truex, D., 2012. An empirical study of IS architectures in French SMEs: integration approaches. Eur. J. Inf. Syst. 21, 287–302.

Boh, W.F., Yellin, D., 2006. Using Enterprise Architecture Standards in Managing Information Technology. J. Manag. Inf. Syst. 23, 163–207.

Boster, M., Liu, S., Thomas, R., 2000. Getting the Most from your Enterprise Architecture. IEEE IT Pro 43–50.

Browning, T.R., 2001. Applying the design structure matrix to system decomposition and integration problems: a review and new directions. IEEE Trans. Eng. Manag. 48, 292–306.

Brøndum, J., Zhu, L., 2010. Towards an Architectural Viewpoint for Systems of Software Intensive Systems Categories and Subject Descriptors. In: Proceedings - International Conference on Software Engineering.

Byrd, T., Turner, D., 2000. Measuring the Flexibility of Information Technology Infrastructure : Exploratory Analysis of a Construct. J. Manag. Inf. Syst. 17, 167–208.

Börjesson, F., 2012. Approaches to Modularity in Product Architecture. Stockholm Royal Institute of Technology.

Börjesson, F., Hölttä-Otto, K., 2012. Improved clustering algorithm for design structure matrix. In: Proceedings of the ASME 2012 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference. pp. 95–109.

Cataldo, M., Wagstrom, P., Herbsleb, J., Carley, K., 2006. Identification of coordination requirements: implications for the Design of collaboration and awareness tools. In: Proceedings of the 2006 Conference on Computer Supported Cooperative Work. pp. 353–362.

Chen, D., Doumeingts, G., Vernadat, F., 2008. Architectures for enterprise integration and interoperability: Past, present and future. Comput. Ind. 59, 647–659.

Danilovic, M., Browning, T.R., 2007. Managing complex product development projects with design structure matrices and domain mapping matrices. Int. J. Proj. Manag. 25, 300–314.

DoD, 2010. The U.S. Department of Defense Architecture Framework Version 2.02 [WWW Document]. URL http://dodcio.defense.gov/dodaf20.aspx (accessed 7.8.13).

Dreyfus, D., 2009. Digital cement: information system architecture, complexity, and flexibility. Boston University.

DSMWeb, n.d. MATLAB Macro for Clustering DSMs [WWW Document]. URL http://www.dsmweb.org/en/dsm-tools/research-tools/matlab.html (accessed 11.8.13).

Duncan, N., 1995. Capturing flexibility of information technology infrastructure: a study of resource characteristics and their measure. J. Manag. Inf. Syst. 12, 37–57.

Eick, S., Graves, T., Karr, A., 2001. Does code decay? Assessing the evidence from change management data. IEEE Trans. Softw. Eng. 27, 1–12.

Eisenhardt, K., 1989. Building Theories from Case Study Research. Acad. Manag. Rev. 14, 532–550.

Eppinger, S.D., Browning, T.R., 2012. Design Structure Matrix Methods and Applications. MIT Press.

Fenton, N., Melton, A., 1990. Deriving Structurally Based Software Measures. J. Syst. Softw. 12, 177–187.

Fernandez, C., 1998. Integration analysis of product architecture to support effective team co-location. Massachusetts Institute of Technology.

Ge, B., Hipel, K.W., Yang, K., Chen, Y., 2013. A Data-Centric Capability-Focused Approach for System-of-Systems Architecture Modeling and Analysis. Syst. Eng. 16, 363–377.

Gershenson, J.K., Prasad, G.J., Zhang, Y., 2003. Product modularity: Definitions and benefits. J. Eng. Des. 14, 295–313.

Gershenson, J.K., Prasad, G.J., Zhang, Y., 2004. Product modularity: measures and design methods. J. Eng. Des. 15, 33–51.

Goldberg, D., Holland, J., 1988. Genetic Algorithms and Machine Learning. Mach. Learn. 3, 95–99.

Hinsman, C., Sangal, N., Stafford, J., 2009. Achieving Agility through Architecture Visibility. In: International Conference on Quality of Software Architecture. pp. 116–129.

Hölttä-Otto, K., Chiriac, N., Lysy, D., Suk Suh, E., 2012. Comparative analysis of coupling modularity metrics. J. Eng. Des. 23, 790–806.

Iacob, M.-E., Jonkers, H., 2013. Quantitative Analysis of Enterprise Architectures. In: Interoperability for Enterprise Software and Applications. pp. 249–262.

Idicula, J., 1995. Planning for concurrent engineering. Singapore.

ISO, 2007. ISO/IEC 42010:2007. Systems and software engineering -- Recommended practice for architectural description of software-intensive systems.

Johnson, P., Ekstedt, M., 2004. Using enterprise architecture for CIO decision-making: On the importance of theory. In: Proceedings of the Second Annual Conference on Systems Engineering Research.

Johnson, P., Johansson, E., Sommestad, T., Ullberg, J., 2007a. A Tool for Enterprise Architecture Analysis. In: 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007). Ieee, pp. 142–156.

Johnson, P., Lagerström, R., Närman, P., Simonsson, M., 2007b. Enterprise architecture analysis with extended influence diagrams. Inf. Syst. Front. 9, 163–180.

Kanniainen, J., 2013. Professor, Tampere University of Technology. Tampere. Interview 7.11.2013.

Lagerström, R., Baldwin, C., MacCormack, A., Aier, S., 2013a. Visualizing and Measuring Enterprise Application Architecture: An Exploratory Telecom Case.

Lagerström, R., Baldwin, C., MacCormack, A., Dreyfus, D., 2013b. Visualizing and Measuring Enterprise Architecture : An Exploratory BioPharma Case. Harvard Bus. Sch. Work. Pap. 13-105.

Lam, W., 2005. Investigating success factors in enterprise application integration: a case-driven analysis. Eur. J. Inf. Syst. 14, 175–187.

Lamantia, M.J., Maccormack, A.D., Rusnak, J., 2007. Evolution Analysis of Large-Scale Software Systems Using Design Structure Matrices & Design Rule Theory.

Langmead, N., 2007. Using DSM to test the software architecture. In: Proceedings of 9th International DSM Conference. pp. 373–382.

Lankhorst et al., M., 2009. Enterprise architecture at work: Modelling, communication and analysis. Springer-Verlag, Berlin Heidelberg.

Lehman, M., 1996. Laws of software evolution revisited. In: Software Process Technology. Springer-Verlag, Berlin, Heidelberg, pp. 108–124.

Lindemann, U., 2009. Building and creating a DSM [WWW Document]. URL http://www.dsmweb.org/en/understand-dsm/technical-dsm-tutorial0/building-creating-dsm.html (accessed 8.4.13).

Lopes, C., Bajracharya, S., 2005. An Analysis Of Modularity In Aspect Oriented Design. In: Proceedings of the 4th International Conference on Aspect-Oriented Software Development. pp. 15–26.

Luftman, J., Ben-Zvi, T., 2012. Key Issues for IT Executives 2011: Cautious optimism in uncertain economic times. MIS Q. Exec. 10, 203–212.

Luftman, J., Kempaiah, R., Nash, E., 2005. Key issues for IT executives 2004. MIS Q. Exec. 4, 269–285.

Luftman, J., Kempaiah, R., Rigoni, E., 2009. Key issues for IT executives 2008. MIS Q. Exec. 8, 151–159.

MacCormack, A., Baldwin, C., Rusnak, J., 2010. The Architecture of Complex Systems : Do Core-periphery Structures Dominate ? Harvard Bus. Sch. Work. Pap. 4770-10.

MacCormack, A., Baldwin, C., Rusnak, J., 2012. Exploring the duality between product and organizational architectures: A test of the “mirroring” hypothesis. Res. Policy 41, 1309–1324.

MacCormack, A., Rusnak, J., Baldwin, C., 2006. Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code. Manage. Sci. 52, 1015–1030.

MacCormack, A., Rusnak, J., Baldwin, C., 2007. The impact of component modularity on design evolution: Evidence from the software industry. Harvard Bus. Sch. Work. Pap. 08-038.

MathWorks, n.d. MATLAB Overview [WWW Document]. URL http://www.mathworks.se/products/matlab/ (accessed 10.31.13).

Mays, N., Pope, C., 1995. Rigour and qualitative research. Bus. Manag. J. 311, 109–112.

Mehta, N., Medvidovic, N., Phadke, S., 2000. Towards a taxonomy of software connectors. In: Proceedings of the 22nd International Conference on Software Engineering. pp. 178–187.

Merriam-Webster, n.d. Component: definition [WWW Document]. URL http://www.merriam-webster.com/dictionary/component (accessed 8.2.13a).

Merriam-Webster, n.d. Hierarchy: definition [WWW Document]. URL http://www.merriam-webster.com/dictionary/hierarchy (accessed 11.10.13b).

Mikaelian, T., Nightingale, D.J., Rhodes, D.H., Hastings, D.E., 2011. Real Options in Enterprise Architecture: A Holistic Mapping of Mechanisms and Types for Uncertainty Management. IEEE Trans. Eng. Manag. 58, 457–470.

Murmann, J., Frenken, K., 2006. Toward a systematic framework for research on dominant designs, technological innovations, and industrial change. Res. Policy 35, 925–952.

Ness, L.R., 2005. Assessing the relationships among IT flexibility, strategic alignment, and IT effectiveness: study overview and findings. J. Inf. Technol. Manag. XVI, 1–17.

Newcomb, P., Bras, B., Rosen, D., 1996. Implications of modularity on product design for the life cycle. In: Proceedings of The 1996 ASME Design Engineering Technical Conferences and Computers in Engineering Conference. pp. 1–12.

Nuutila, E., 1995. Efficient Transitive Closure Computation in Large Digraphs. The Finnish Academy of Technology.

Offutt, A., Harrold, M., Kolte, P., 1993. A software metric system for module coupling. J. Syst. Softw. 20, 295–308.

Ota, D., Gerz, M., 2011. Benefits and Challenges of Architecture Frameworks. In: 16th International Command and Control Research and Technology Symposium.

Papazoglou, M., Heuvel, W.-J., 2007. Service oriented architectures: approaches, technologies and research issues. VLDB J. 16, 389–415.

Parnas, D., 1972. On the criteria to be used in decomposing systems into modules. Commun. ACM 15, 1053–1058.

Pimmler, T., Eppinger, S., 1994. Integration analysis of product decompositions. In: Proceedings of the ASME Sixth International Conference on Design Theory and Methodology.

Porter, M., Millar, V., 1985. How information gives you competitive advantage. Harv. Bus. Rev. 149–174.

Pouloudi, A., 1999. Aspects of the Stakeholder Concept and their Implications for Information Systems Development. In: Proceedings of the 32nd Hawaii International Conference on System Sciences. pp. 1–17.

Richardson, G., Jackson, B., Dickson, G., 1990. A Principles-Based Enterprise Architecture : Lessons From Texaco and Star Enterprise. MIS Q. 14, 385–403.

Roosmalen, H. Van, 2007. Assessment and improvement of software systems by applying DSM. In: Proceedings of 9th International DSM Conference. pp. 325–336.

Sangal, N., Jordan, E., Sinha, V., Jackson, D., 2005. Using dependency models to manage complex software architecture. In: Proceedings of the 20th Annual ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications. ACM Press, New York, New York, USA, pp. 167–176.

Saunders, M., Lewis, P., Thornhill, A., 2009. Research methods for business students, 5th ed. Pearson Education Limited, Essex.

Schach, S., Jin, B., Wright, D., Heller, G., Offutt, A., 2002. Maintainability of the Linux kernel. IEE Proceedings-Software 149, 18–23.

Schekkerman, J., 2005. Trends in Enterprise Architecture 2005.

Schilling, M., 2000. Toward a General Modular Systems Theory and Its Application to Interfirm Product Modularity. Acad. Manag. Rev. 25, 312–334.

Schmidt, C., Buxmann, P., 2010. Outcomes and success factors of enterprise IT architecture management: empirical insight from the international financial services industry. Eur. J. Inf. Syst. 20, 168–185.

Sharman, D., Yassine, A., 2004. Characterizing complex product architectures. Syst. Eng. 7, 35–60.

Sharman, D., Yassine, A., 2007. Architectural Valuation using the Design Structure Matrix and Real Options Theory. Concurr. Eng. 15, 157–173.

Siggelkow, N., 2007. Persuasion with case studies. Acad. Manag. J. 50, 20–24.

Simon, H., 1962. The Architecture of Complexity. Proc. Am. Philos. Soc. 106, 467–482.

Simpson, J., Simpson, M., 2009. System of systems complexity identification and control. In: System of Systems Engineering, 2009. SoSE 2009. IEEE International Conference On. pp. 1–6.

Sosa, M., Browning, T., Mihm, J., 2007a. Studying the dynamics of the architecture of software products. In: Proceedings of the ASME 2007 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference. pp. 1–14.

Sosa, M., Eppinger, S., Rowles, C., 2003. Identifying Modular and Integrative Systems and Their Impact on Design Team Interactions. J. Mech. Des. 125, 240–252.

Sosa, M., Eppinger, S., Rowles, C., 2004. The Misalignment of Product Architecture and Organizational Structure in Complex Product Development. Manage. Sci. 50, 1674–1689.

Sosa, M., Eppinger, S., Rowles, C., 2007b. A Network Approach to Define Modularity of Components in Complex Products. J. Mech. Des. 129, 1118–1129.

Sowa, J., Zachman, J., 1992. Extending and formalizing the framework for information systems architecture. IBM Syst. J. 31, 590–616.

Stelzer, D., 2009. Enterprise architecture principles: literature review and research directions. In: Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops. pp. 12–21.

Steward, D., 1981. The design structure system: a method for managing the design of complex systems. IEEE Trans. Eng. Manag. 28, 9–12.

Stevens, S.S., 1946. On the Theory of Scales of Measurement. Science (80-.). 103, 677–80.

Sullivan, K., Griswold, W., Cai, Y., Hallen, B., 2001. The structure and value of modularity in software design. In: ACM SIGSOFT Software Engineering Notes. pp. 99–108.

Tamm, T., Seddon, P., Shanks, G., Reynolds, P., 2011. How does enterprise architecture add value to organisations? Commun. Assoc. Inf. Syst. 28, 141–168.

Tang, A., Han, J., Chen, P., 2004. A Comparative Analysis of Architecture Frameworks.

Tanriverdi, H., 2005. Information Technology Relatedness, Knowledge Management Capability, and Performance of Multibusiness Firms. MIS Q. 29, 311–334.

The Open Group, 2011. TOGAF Version 9.1, 1st ed. Van Haren Publishing, Zaltbommel.

Thebeau, R., 2001. Knowledge Management of System Interfaces and Interactions for Product Development Processes. Massachusetts Institute of Technology.

Tushman, M., Murmann, J., 1998. Dominant designs, technology cycles and organizational outcomes. Res. Organ. Behav. 20, 231–266.

Waldman, F., Sangal, N., 2007. Applying DSM to enterprise architectures. In: Proceedings of 9th International DSM Conference. pp. 61–71.

Warfield, J.N., 1973. Binary Matrices in System Modeling. IEEE Trans. Syst. Manag. Cybern. 3, 441–449.

Webster, J., Watson, R., 2002. Analyzing the Past to Prepare for the Future: Writing a Literature Review. MIS Q. 26, xiii–xxiii.

Winter, R., Fischer, R., 2007. Essential layers, artifacts, and dependencies of enterprise architecture. J. Enterp. Archit. 1–12.

Xia, F., 2000. On the concept of coupling, its modeling and measurement. J. Syst. Softw. 50, 75–84.

Yin, R., 2003. Case Study Research: Design and Method, 3rd ed. Sage, London.

Yu, T.-L., Yassine, A., Goldberg, D., 2007. An information theoretic method for developing modular architectures using genetic algorithms. Res. Eng. Des. 18, 91–109.

Zachman, J., 1987. A framework for information systems architecture. IBM Syst. J. 26, 276–292.

Zachman, J., n.d. Architecture is Architecture is Architecture [WWW Document]. URL http://www.zachman.com/ea-articles-reference/52-architecture-is-architecture-is-architecture-by-john-a-zachman (accessed 8.22.13).

Zakarian, A., 2008. A New Nonbinary Matrix Clustering Algorithm for Development of System Architectures. IEEE Trans. Syst. Man, Cybern. Part C (Applications Rev. 38, 135–141.

APPENDIX 1: Leading IS journals

According to the Association of Information Systems, the leading IS journals are, in alphabetical order:

- European Journal of Information Systems
- Information Systems Journal
- Information Systems Research
- Journal of AIS
- Journal of Information Technology
- Journal of MIS
- Journal of Strategic Information Systems
- MIS Quarterly

APPENDIX 2: Search strings used in literature review for DSM in EA context

Scopus:

(TITLE-ABS-KEY("enterprise architecture" OR "application architecture" OR "IT architecture" OR "service-oriented architecture" OR "information systems architecture" OR "data architecture" OR "technology architecture" OR "system-of-systems" OR "system of systems") AND TITLE-ABS-KEY("dsm" OR "design structure matrix" OR "dependency structure matrix" OR "dependency structure method" OR "dependency source matrix" OR "dependency system model" OR "deliverable source map" OR "problem solving matrix" OR "incidence matrix" OR "N^2 matrix" OR "N squared matrix" OR "interaction matrix" OR "dependency map"))

WoK:

Topic=("enterprise architecture" OR "application architecture" OR "IT architecture" OR "service-oriented architecture" OR "information systems architecture" OR "data architecture" OR "technology architecture" OR "system-of-systems" OR "system of systems") AND Topic=("dsm" OR "design structure matrix" OR "dependency structure matrix" OR "dependency structure method" OR "dependency source matrix" OR "dependency system model" OR "deliverable source map" OR "problem solving matrix" OR "incidence matrix" OR "N^2 matrix" OR "N squared matrix" OR "interaction matrix" OR "dependency map")

APPENDIX 3: Explanations of variables in the objective function of the IGTA algorithm

Variable	Explanation
TotalCost	The final result of the calculation.
IntraClusterCost	The cost of connections within clusters.
ExtraClusterCost	The cost of connections outside clusters.
DSM(i,j)	The value in the DSM at cell (i,j).
ClusterSize_y	The number of elements in cluster "y".
DSMSize	The total number of elements in the DSM.
powcc	Exponent given as parameter to penalize the size of clusters.