Enterprise Systems Modifiability Analysis

An Enterprise Architecture Modeling Approach for Decision Making

ROBERT LAGERSTRÖM

Doctoral Thesis
Stockholm, Sweden 2010
Abstract

Contemporary enterprises depend to a great extent on software systems. During the past decades the number of systems has been constantly increasing and these systems have become more integrated with one another. This has lead to a growing complexity in managing software systems and their environment. At the same time business environments today need to progress and change rapidly to keep up with evolving markets. As the business processes change, the systems need to be modified in order to continue supporting the processes. The complexity increase and growing demand for rapid change makes the management of enterprise systems a very important issue. In order to achieve effective and efficient management, it is essential to be able to analyze the system modifiability (i.e., estimate the future change cost). This is addressed in the thesis by employing architectural models.

The contribution of this thesis is a method for software system modifiability analysis using enterprise architecture models. The contribution includes an enterprise architecture analysis formalism, a modifiability metamodel (i.e., a modeling language), and a method for creating metamodels. The proposed approach allows IT-decision makers to model and analyze change projects. By doing so, high-quality decision support regarding change project costs is received.

This thesis is a composite thesis consisting of five papers and an introduction. Paper A evaluates a number of analysis formalisms and proposes extended influence diagrams to be employed for enterprise architecture analysis. Paper B presents the first version of the modifiability metamodel. In Paper C, a method for creating enterprise architecture metamodels is proposed. This method aims to be general, i.e., can be employed for other IT-related quality analyses such as interoperability, security, and availability. The paper does however use modifiability as a running case. The second version of the modifiability metamodel for change project cost estimation is fully described in Paper D. Finally, Paper E validates the proposed method and metamodel by surveying 110 experts and studying 21 change projects at four large Nordic companies. The validation indicates that the method and metamodel are useful, contain the right set of elements and provide good estimation capabilities.

Keywords: Enterprise Architecture, Software System Modifiability, Decision Making, Metamodeling, Enterprise Architecture Analysis, Software Change Cost Estimation
Acknowledgments

Prof. Pontus Johnson, Prof. Torsten Cegrell and Judith Westerlund.

Susanne Rautiainen, Minna Lagerström, Anette Lagerström, Bertil Lagerström, Tobias Lagerström, Maj Lagerström, Harriet Johnson and Berth Johnson.

Mathias Ekstedt, Joakim Lilliesköld, Lars Nordström, Per Närman, Pia Gustafsson, Moustafa Che-nine, Johan Ullberg, Teodor Sommestad, Ulrik Franke, David Höök, Johan König, Kun Zhu, Evelina Ericsson, Waldo Rocha Flores, Markus Buschle, Liv Marcks von Würtemberg, Hannes Holm, Erik Johansson, Magnus Gammelgård, Märten Simonsson, Jan Saat and Stephan Aier.

Mårten Edgren, Stefan Jonsson, Tobias Gustafsson, Najib Mirkhani, Kennet Namini and Claes Holmström.

Robert Diggs

Thank you!

Stockholm, February 2010

Robert Lagerström
Papers

List of included papers


Publications not included in the thesis


Publication XVII: Johan Ullberg, Robert Lagerström, and Pontus Johnson, "Enterprise Architec-


# Table of Contents

## I Introduction

1 Introduction .......................... 1

1.1 Outline of the thesis .......................... 3
1.2 Background .......................... 3
1.3 Research purpose .......................... 5
1.4 Related work .......................... 6
1.5 Results .......................... 8
1.6 Contribution .......................... 12
1.7 Research design .......................... 14

## Bibliography

17

## II Papers A to E

2 Paper A: Enterprise Architecture Analysis with Extended Influence Diagrams .......................... 33

3 Paper B: Analyzing System Maintainability using Enterprise Architecture Models .......................... 53


6 Paper E: Architecture Analysis of Enterprise Systems Modifiability – Models, Analysis, and Validation .......................... 133
Part I

Introduction
Chapter 1

Introduction

1.1 Outline of the thesis

This thesis is divided into two parts. The first part is an introduction describing and summarizing the second part which contains the five papers constituting the core of the thesis. The introduction includes the background of the presented work, the research purpose addressed in this thesis, related works, a summary of the main results in terms of a method for software system modifiability analysis using enterprise architecture models, the main research contributions, and a depiction of the research design. The five papers (Paper A to Paper E) of part two in this thesis describe the details of the research and results. The papers have either been published (A–C), submitted (D) or accepted (E) in academic journals.

1.2 Background

Contemporary enterprises depend to great extent on software systems. During the past decades the number of systems has constantly been increasing and these systems have become more integrated with one another. Due to the growing number of systems and their tight coupling, management of software systems and their environment has become a complex business. In order to achieve effective and efficient management of the enterprise systems it is essential to be able to analyze the current status of system qualities such as availability, performance, security, and modifiability, as well as estimate their status in different future scenarios. Estimation of these qualities is a great challenge that to a large extent can be addressed by introducing architectural models as a means of abstraction.

Enterprise system modifiability

Business environments today need to progress and change rapidly to keep up with evolving markets. Most business processes are supported by software systems and as the business processes change, the systems need to be modified in order to continue supporting the processes. Modifications include extending, deleting, adapting, and restructuring the enterprise software systems [1]. Modification efforts can range from adding a functional requirement in a single system to implementing a service-oriented architecture for a complete enterprise.

An essential issue with today’s software systems is that many of them are interconnected, thus a modification to one system may cause a ripple effect among other systems. It is also common that the systems have been developed and modified during many years. Making further changes to these systems often require a lot of effort from the organization, for instance due to a large
CHAPTER 1. INTRODUCTION

amount of previous modifications implemented ad hoc. Problems like these raise questions for IT-
decision makers such as: Is there enough documentation describing the system? Is the source code
easy to understand? Is the change management process mature enough? Or, which systems are
interconnected?

According to IEEE’s Standard Glossary of Software Engineering [2] maintainability is defined
as the ease with which a system can be modified and maintenance is defined as the process of
modifying a system or component. The terms maintainability and modifiability are therefore used
interchangeably throughout the thesis.

Several studies indicate that in the lifecycle of an enterprise-wide system the modification work
consumes the greatest portion of resources: Harrison and Cook report that over 70 % of the software
budget is spent on maintenance [3], Pigoski refers to studies stating that the maintenance cost,
relative to the total lifecycle cost of a system, has been increasing from 40 % in the early 1970s
up to 90 % in the early 1990s [4], and Jarzabek states that “the cost of maintenance, rather than
dropping, is on the increase” [5].

The activities of modifying enterprise systems are typically executed in projects, and IT-decision
makers often find it difficult to estimate and plan their change projects. Thus, a large proportion of
the projects aiming to modify a software system environment fail, i.e. the projects tend to take longer
time and cost more than expected. Laird and Brennan declares that 23 % of all software projects
are cancelled before completion, whereas of those completed only 28 % are delivered on time, and
the average software project exceeds its budget by 45 % [6]. This can often occur due to lack of
information about the systems being changed. According to Laird and Brennan, software engineers
must be able to understand their activities, as well as manage the risks, through estimation and
measurement [6]. Therefore, it would be useful for IT-decision makers to gather more information
in a structured manner and use this information to analyze how much effort a certain modification
of an enterprise-wide system would require.

This thesis will address these issues of change, i.e. analyzing enterprise system modifiability,
by employing architecture modeling.

Enterprise architecture

In recent years, Enterprise Architecture (EA) has become an established discipline for both business
and software system management [7]. EA describes the fundamental artifacts of business and IT as
well as their interrelationships in a single aggregate model. Architecture models constitute the core
of the approach and serve the purpose of making the complexities of the real world understandable
and manageable [8]. Architecture models have three main functions [9]: 1) documentation, 2)
analysis, 3) planning and design of the enterprise architecture. These three purposes, providing
decision support for the stakeholders, are in turn crucial to the success of management tasks such
as IT/business alignment, product planning, IT-governance, business development, and business
process consolidation [10].

There are many reasons why enterprise architecture is suitable for modifiability analysis. The
three most important issues are: Firstly, large contemporary enterprises do not have ten systems to
modify. They have hundreds or even thousands of systems. Secondly, there are only few standalone
applications in the modern software system landscape. Many systems are tightly integrated both
with each other and with the business processes they support. Thirdly, modification work is no
longer an isolated task of one or few engineers. The complex changes being implemented involve
business executives, project managers, architects, developers, testers etc.

The purpose of creating enterprise architecture models and conducting analyses of these is thus
to facilitate the making of rational decisions about software systems and business related issues
in an organization. However, what constitutes a “good” enterprise architecture model has thus far
not been clearly defined. This is due to the fact that the "goodness" of the model is not an inherent
property, but contingent on the purpose the model is intended to fill, i.e. what kind of analyses it will be subjected to [11, 12]. For instance, if an enterprise architecture model is employed to evaluate the modifiability of a software system, the information required from the model differs radically from the case when the model is used to evaluate the system’s usability. Thus, a metamodel focusing on modifiability analysis needs to be tailored for that purpose.

Decision making, whether the decisions apply to software systems or not, is rarely performed under conditions of complete certainty. Therefore, when tailoring a metamodel for analysis various uncertainties should be considered. There is uncertainty regarding how the world actually behaves, i.e. knowledge of exactly how various phenomena affect one another is seldom certain (causal uncertainty). An example of this would be uncertainty with respect to the causal effect the size of a system may have on the cost of software change. Being able to formally handle causal uncertainty among the criteria studied allows the approach to estimate and predict future actions regarding qualities such as modifiability. Another fundamental uncertainty regards the definitions of various concepts. One example is the notion of software complexity, that might be interpreted as for instance Halstead complexity, cyclomatic complexity, or simply lines of code. Since there is no general agreement on the meaning of all concepts (definitional uncertainty), it is important that the approach is explicit about the intended meaning of the terms used. Besides the handling of uncertainties, an advantage of the approach would if it could express goals and decisions of the stakeholders, thereby making the selection of decision alternatives easier since the stakeholder can express the domain of control and its effect on the chosen goal. Another benefit of the approach would be if it could provide a quantitative assessment, making it easier for the decision maker to choose between different decision scenarios. Finally, if the approach could be coupled with an inference engine providing the quantitative assessment, handling the uncertainties and the causality formalization, this would be of great use.

1.3 Research purpose

The purpose of the research presented in this thesis is to develop a method that provides decision support for system modifiability using enterprise architecture models. More specifically the method should permit modeling of enterprise-wide software systems and their environment, as well as enabling estimation of software system modifiability. The main purpose is therefore:

- To develop a method for system modifiability analysis using enterprise architecture models.

In order to be able to employ such a method the enterprise architecture analysis approach needs to be formalized, a modifiability metamodel supporting modification decision making needs to be designed and validated, and in order to create a suitable metamodel a method for metamodel creation needs to be developed and validated (cf. Fig. 1.1). Thus, the underlying goals to be addressed in order to fulfill the main purpose are:

1. To develop an enterprise architecture analysis approach which: takes uncertainties into consideration, provides a quantitative assessment, expresses quality criteria in an unambiguous manner and is coupled with an inference engine.

2. To develop an enterprise architecture metamodel creation method that: focuses on the design of metamodels for quantitative analysis under uncertainty, helps the metamodeler to choose the appropriate criteria for the metamodel and in a structured way describe how these criteria relate to each other and the quality goal chosen.

3. To develop an enterprise architecture metamodel for system modifiability that: contains the appropriate criteria for modifiability analysis and has the ability to estimate the cost of change projects.
4. To test and validate the proposed method with expert data and case studies of change projects.

![Diagram showing main purpose and underlying goals](image)

**Figure 1.1:** The main purpose and the four underlying goals to be addressed in this thesis.

### 1.4 Related work

In developing the method for system modifiability analysis using enterprise architecture models, two main research disciplines were covered: software system modifiability and enterprise architecture. This section of the thesis presents the related work within these two disciplines.

#### Software system modifiability

The issue of dealing with modifiability is not an enterprise architecture specific problem. Managing and assessing system change has been addressed in research for many years. Several authors have contributed to the understanding of modifiability including [1, 3, 4, 5, 6, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]. Some of the more well-known assessment approaches include the COnstructive COst MOdel (COCOMO), the Software Architecture Analysis Method (SAAM), the Oman taxonomy, and the ISO/IEC 9126 standard. These are briefly described below.

COCOMO, COnstructive COst MOdel, was in its first version released in the early 1980’s. It became one of the most frequently used and most appreciated software cost estimation models of that time. Since then, development and modifications of COCOMO have been performed several times to keep the model up to date with the continuously evolving software development trends. The latest version of COCOMO, called COCOMO II, had its estimation capabilities calibrated in the year 2000 with the help of information from 161 project data points and 8 experts. This latest calibrated version of COCOMO II uses the probabilistic Bayesian approach for turning a priori obtainable data into estimates of costs related to an a posteriori state of a software development or modification project [13, 24, 32, 33, 101, 102, 103, 104, 105, 106, 107, 108, 109].

Bass et al. proposes the Software Architecture Analysis Method (SAAM) for software quality evaluation [1, 110, 111]. This method takes several quality attributes into consideration; performance, security, availability, functionality, usability, portability, reusability, testability, integrability


and modifiability. Bass et al. categorizes modifications as: extending or changing capabilities, deleting unwanted capabilities, adapting to new operating environments and restructuring. Based on the quality attributes presented, Bass et al. propose different architectural styles which then are employed in the SAAM. SAAM is a scenario-based approach which intends to make sure that stakeholder quality goals are met (for instance high modifiability). According to Bass et al. SAAM can be used in two contexts: as a validation step for an architecture being developed or as a step in the acquisition of a software system. Besides SAAM there are several other methods supporting analysis of software architecture quality attributes, such as Architecture Trade-off Analysis Method (ATAM) [112, 113], Cost Benefit Analysis Method (CBAM) [60], Architecture Level Modifiability Analysis (ALMA) [21, 114, 115, 116], and Aspectual Software Architecture Analysis Method (ASAAM) [117].

The Definition and Taxonomy for Software Maintainability presented by Oman et al. in 1992 provides a hierarchical definition of software maintainability in the form of a taxonomy [79, 118]. Oman et al. found three broad categories of factors influencing the maintainability of a software system; management, operational environment, and the target software system. Each of these top-level categories is then further broken down into measurable attributes. According to Oman et al. the taxonomy can be useful for developers by defining characteristics affecting the software maintenance cost of the software they are developing. Hence, the developers can write highly maintainable software from the beginning by studying the taxonomy. Maintenance personnel can use the taxonomy to evaluate the maintainability of the software they are working with in order to pin point risks etc. Project managers and architects can use the taxonomy in order to prioritize projects and locate areas in need of re-design.

ISO/IEC 9126 is an international standard for software engineering focusing on software quality [55, 119, 120, 121]. The proposed quality model contains six quality attributes; functionality, reliability, usability, efficiency, maintainability, and portability. The aim with this quality model is to provide definitions of the quality attributes and provide a set of sub-characteristics that influence these quality attributes. ISO/IEC defines maintainability as; “the capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications”. Maintainability is divided into analyzability, changeability, stability, and testability. For each of these sub-characteristics ISO/IEC provides a set of metrics for evaluation. According to ISO/IEC the quality models can be used to validate the completeness of a requirements definition, identify software requirements, identify software design objectives, identify software testing objectives, identify quality assurance criteria, and identify acceptance criteria for a completed software product.

The available methods for modifiability analysis are not focusing on change in an enterprise architecture context. There are many problems that need to be addressed that the available methods miss, such as: the increasing number of systems affected by enterprise-wide changes, the tight integration between systems, the increasing involvement of diverse people in a company e.g. business executives, project managers, architects, developers, testers. Some methods do use models, other employ quality criteria, some has a formal analysis engine, and there are methods using scenarios in decision making situations. There is however no method having brought it all together in an EA context. The studied methods provided valuable input for the EA approach presented in this thesis.

Enterprise architecture

Enterprise Architecture (EA) has grown into a discipline widely recognized with many initiatives. In the discipline of EA, methods and tools for management of the complex combinations of IT-systems in modern organizations are found. Many consider John Zachman’s article A Framework for Information Systems Architecture [122] to be the starting point of EA. Since then, many authors have contributed to the field, for instance in EA books such as [7, 11, 123, 124, 125, 126, 127, 128,
CHAPTER 1. INTRODUCTION

129, 130, 131, 132, 133, 134], in doctoral theses such as [135, 136, 137, 138, 139, 140, 141, 142], in articles such as [8, 9, 10, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176], with frameworks such as DoDAF [177], EAF2 [178], E2AF [179], FEA [180], GERAM [181], IAF [182], MoDAF [183], NAF [184], TOGAF [185], and with tools such as Adaptive Enterprise Architecture Manager [186], ADOit [187], ARIS [188, 189], EAT [190, 191], EA/Studio [192], planningIT [193], System Architect [194], and Troux [195].

Many EA initiatives tend to focus on the differentiation of metamodels in viewpoints, often referred to as architecture layers. Neither the larger frameworks nor the metamodel oriented initiatives typically contain metamodels suitable for specific analyses (except on a very general level). Only a few metamodels detail whether and how they support decision making. General and notation-independent methods for metamodeling that support decision making and analysis are rare. None of the examined methods, frameworks, and notations are analysis-oriented, results in metamodels and covers the entire EA domain. Furthermore, there is little support of quality criteria formalization in the available methods for enterprise metamodel creation. Although some EA initiatives provide some guidance to security analysis [196, 197] or availability and performance analysis [125] no EA approaches found offer architecture models for software system modifiability analysis in a formalized way.

Since there seem to be no EA approaches focusing on modifiability analysis in a formalized manner this thesis aims at providing such a formalized method.

1.5 Results

The main result presented in this thesis is a method for enterprise system modifiability analysis using enterprise architecture models. Within this method there are four important deliverables. The first deliverable is an enterprise architecture analysis approach. Deliverable two is a method for creating enterprise architecture metamodels for analysis. The third deliverable is an enterprise architecture metamodel for modifiability analysis. Deliverable four is a validation of the method.

The method for system modifiability analysis using enterprise architecture modeling

Given that an IT-decision maker desires to analyze system modifiability in order to get better decision support, four steps need to be considered.

1. Create models. The first step concerns the development of the models describing the change to be implemented and the surrounding environment. The modifiability metamodel guides the modeler in what to model.

2. Analyze modifiability. Step two focuses on the analysis of the modeled scenario using the inference engine provided in the presented EA approach.

3. Make decision. As a third step a decision needs to be made whether to change the scenario and update the models or to proceed with the modeled scenario as it is.

4. Implement change. When the decision maker is satisfied with the modeled scenario and the analysis the modification needs to be implemented in the organization, i.e. it is time to start the change project.

Fig. 1.2 presents the EA analysis concept. In order to employ this approach for decision making the enterprise architecture analysis approach needs to be formalized, a modifiability metamodel supporting modification decision making needs to be designed and validated, and in order to create
1.5. RESULTS

A suitable metamodel a method for metamodel creation needs to be developed and validated. These deliverables are presented in the subsequent sections.

![Diagram of Enterprise Architecture Models, Analysis, and Analysis Results]

Figure 1.2: Enterprise architecture models are subjected to analysis in order to assess various qualities e.g. security, availability, and modifiability.

**Enterprise architecture analysis**

In the proposed method for system modifiability analysis an important issue is the analysis formalism supporting the method. This thesis presents the use of a formal language to support EA analysis. As discussed in the background and in Paper A, such a language needs to be able to represent causal relations between, and definitions of, various criteria as well as uncertainty with respect to both criteria and relations. To support decision making properly, the language must also allow the representation of goals and decision alternatives.

Paper A evaluates a number of languages with respect to these requirements, and selects influence diagrams for further consideration. The influence diagrams are then extended to fully satisfy the requirements. The syntax and semantics of the extended influence diagrams (EIDs) are detailed in Paper A, and their use is demonstrated in an example. Fig. 1.3 illustrates the relation between an enterprise architecture metamodel and an extended influence diagram. Paper B presents the first version of the modifiability metamodel which is based on the extended influence diagram approach, cf. Fig. 1.5.

**Method for creating enterprise architecture metamodels**

In the proposed method for system modifiability analysis, enterprise architecture models are used in order to increase the understanding of enterprise systems and specifically to perform modifiability analysis. To be able to employ such analysis, a metamodel containing the appropriate criteria and their relationships needs to be created. As argued for in the thesis background and in Paper C, a creation method for analysis centered metamodels must take several issues into consideration in order to come up with a suitable metamodel. Firstly, the method should consider correlation and causality in order to support estimation and decision making. Secondly, it should cope with uncertainties such as definitional uncertainty, theoretical heterogeneity, and causal uncertainty.
CHAPTER 1. INTRODUCTION

Figure 1.3: The link between an enterprise architecture model and its associated analysis language.

Paper C describes a method for creating metamodels for EA analysis, cf. Fig. 1.4 for a short version of the method including examples. The proposed method for creating metamodels is general, however Paper C presents the method by creating a metamodel for system modifiability (i.e. the cost of making changes to enterprise-wide systems). The method is validated based on data from 110 experts gathered in surveys and workshops. The usability of the method is shown by referring to six other metamodels developed with the method in projects focusing on other analyses than modifiability, e.g. interoperability, business value, system quality, and dependency analysis.

Figure 1.4: The iterative goal break-down process, a resulting goal break-down example, and the corresponding metamodel.

The modifiability metamodel

This thesis proposes enterprise architecture models as the core of the approach for modifiability analysis. Therefore, an important part of the presented method is the modifiability metamodel. As presented in the background section and in Paper B and D such a metamodel must contain the appropriate criteria for modifiability analysis. It also needs to be designed for analysis purposes supporting decision making. In this thesis the focus has been on estimating change project cost.
1.5. RESULTS

Paper B presents a metamodel coupled with an extended influence diagram for modifiability analysis, cf. Fig. 1.5. This is the first version of the modifiability metamodel.

![Extended influence diagram for modifiability analysis](image)

![Metamodel for modifiability analysis](image)

Figure 1.5: The first version of the modifiability metamodel and the accompanying extended influence diagram.

After method improvements, literature studies, and case studies a second version of the modifiability metamodel was developed. Paper D presents this second version of the metamodel for enterprise systems modifiability analysis, i.e. assessing the cost of making changes to enterprise-wide systems. Fig. 1.6 presents a high level view of the metamodel. The enterprise architecture models in Paper D are formalized using probabilistic relational models (PRMs) enabling the combination of regular entity-relationship modeling aspects with means to perform enterprise architecture analysis [198]. The EA approach with extended influence diagrams coupled with a metamodel is similar to the PRM approach. The main differences are that the EID version formally contains goals, decisions, and definitions, while the PRM version handles multiplicities in a more formal way.

**Method and metamodel validation**

The method for system modifiability analysis using enterprise architecture modeling was tested and validated with experts and case studies. Paper E presents instantiated architectural models based on the metamodel for enterprise systems modifiability analysis. These instantiated models are based on 21 software change projects conducted at four large Nordic companies. The modifiability metamodel employed in the analysis is validated with survey and workshop data from 110 experts and with the data collected in the 21 software change projects. The validation indicates that the modifi-
ability metamodel contains the appropriate set of elements. By studying the estimation accuracy it also indicates that the metamodel produces good estimates, cf. Table 1.1. These estimates are compared with other well known assessment methods such as COCOMO and Function Points indicating that the presented method is a suitable approach for modifiability analysis.

1.6 Contribution

This section summarizes the main contributions of this thesis. Being a composite thesis, the details of the contributions can be found in the included papers, Paper A to E, in the second part of the thesis.

- **The method for modifiability analysis using enterprise architecture models.** The overall method is presented in section 1.5 Results and it pervades all five papers included in this thesis. Basically the method utilizes the rest of the contributions listed below, and explains how the enterprise architecture analysis approach, the metamodel creation method, and the modifiability metamodel can support decision making. When decision makers employ the method, enterprise architecture models are created and analyzed.

- **The enterprise architecture analysis approach.** The approach presented in Paper A contains a formal language coupled with enterprise architecture models providing analysis support of various qualities such as modifiability. The proposed formal language is able to represent causal relations between, and definitions of, various criteria as well as uncertainty related to both the criteria and their relations. The language also allows representation of goals and decision alternatives.
### 1.6. CONTRIBUTION

<table>
<thead>
<tr>
<th>Project size</th>
<th>Project</th>
<th>Actual cost (man-hours)</th>
<th>Estimated cost (man-hours)</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large</td>
<td>A</td>
<td>20 000</td>
<td>19 860</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>20 000</td>
<td>15 230</td>
<td>0.76</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>14 000</td>
<td>14 940</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>9 100</td>
<td>9 580</td>
<td>0.95</td>
</tr>
<tr>
<td>Medium</td>
<td>J</td>
<td>6 266</td>
<td>5 920</td>
<td>0.94</td>
</tr>
<tr>
<td></td>
<td>H</td>
<td>6 228</td>
<td>5 840</td>
<td>0.94</td>
</tr>
<tr>
<td></td>
<td>M</td>
<td>5 810</td>
<td>5 510</td>
<td>0.95</td>
</tr>
<tr>
<td></td>
<td>P</td>
<td>4 458</td>
<td>6 380</td>
<td>0.57</td>
</tr>
<tr>
<td></td>
<td>U</td>
<td>3 595</td>
<td>4 110</td>
<td>0.86</td>
</tr>
<tr>
<td>Small</td>
<td>L</td>
<td>3 200</td>
<td>3 085</td>
<td>0.96</td>
</tr>
<tr>
<td></td>
<td>K</td>
<td>3 200</td>
<td>2 195</td>
<td>0.69</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>3 000</td>
<td>2 480</td>
<td>0.83</td>
</tr>
<tr>
<td></td>
<td>I</td>
<td>2 440</td>
<td>2 170</td>
<td>0.89</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>2 300</td>
<td>2 240</td>
<td>0.97</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>2 054</td>
<td>1 915</td>
<td>0.93</td>
</tr>
<tr>
<td>Small (≤ 200 man-hours or less)</td>
<td>G</td>
<td>1 200</td>
<td>2 095</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>1 082</td>
<td>1 680</td>
<td>0.45</td>
</tr>
<tr>
<td></td>
<td>O</td>
<td>952</td>
<td>2 040</td>
<td>&lt;0</td>
</tr>
<tr>
<td></td>
<td>N</td>
<td>894</td>
<td>2 295</td>
<td>&lt;0</td>
</tr>
<tr>
<td></td>
<td>Q</td>
<td>454</td>
<td>2 005</td>
<td>&lt;0</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>262</td>
<td>1 805</td>
<td>&lt;0</td>
</tr>
</tbody>
</table>

Table 1.1: The method seems to provide accurate modifiability assessments for change projects large than 2000 man-hours.

- **The metamodel creation method for enterprise architecture analysis.** The creation method outcome is an enterprise architecture metamodel for analysis of any chosen quality, e.g. modifiability. The method considers correlation and causality in order to support estimation and decision making. It also copes with uncertainties such as definitional uncertainty, causal uncertainty, and theoretical heterogeneity. The method is mainly described in Paper C.

- **The enterprise architecture metamodel for modifiability analysis.** The metamodel combines the discipline of enterprise architecture and its principles of model-based decision making with content from the field of software system modifiability assessment. The first version of the metamodel is presented in Paper B and the second version is described in Paper D. The second version defines modifiability in terms of software change project cost measured in man-hours. The metamodel contains aspects such as system size, component complexity, documentation quality, change difficulty, and team expertise.

- **Testing and validating the method and metamodel.** The method and metamodel are tested and validated with data from 21 software change projects studied at four large Nordic companies and with survey and workshop data from both academic and industrial experts. Four questions are considered: 1) Are there criteria missing in the metamodel? Answers from 73 experts indicate that there is no important criteria missing in the metamodel. 2) Does the metamodel contain criteria that can be removed? Data from 83 experts indicate that there is no criteria that should be removed from the metamodel. 3) Does the metamodel provide good estimation capabilities? The information gathered in the 21 change projects indicate that the metamodel provides accurate estimations. 4) Is the usability of the creation method high? The creation method proposed has been used in six other metamodel creation projects focusing...
on other quality goals than modifiability, thus indicating that the method is not only usable in the modifiability case. Questions 1 to 3 are mainly addressed in Paper E and question 4 is considered in Paper C.

1.7 Research design

This section covers the methodological aspects that have guided the work in this thesis. Details of how the research has been conducted can be found in Papers A to E.

Research strategies

There are numerous ways to conduct research. Yin [199] discusses five different research strategies and in what situations these are suitable, cf. Table 1.2. The different strategies vary with what type of research question the investigator aims at answering, the extent of control the investigators has over the events studied, and if the focus is of contemporary events or not.

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Form of research question</th>
<th>Requires control of behavioral events?</th>
<th>Focuses on contemporary events?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment</td>
<td>how, why?</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Survey</td>
<td>who, what, where, how many, how much?</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Archival analysis</td>
<td>who, what, where, how many, how much?</td>
<td>no</td>
<td>yes/no</td>
</tr>
<tr>
<td>History</td>
<td>how, why?</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Case study</td>
<td>how, why?</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 1.2: Relevant situations for different research strategies according to Yin.

In this thesis, surveys were used in the phase of creating the modifiability metamodel. In the latter phase, when the method and metamodel were tested and validated in change projects, case studies were employed.

Data collection

In each phase of the research work, different data collection methods were employed. Table 1.3 illustrates when each approach was used. The subsequent paragraphs describe the strengths and weaknesses for the employed data collection methods.

Literature studies were used when developing the EA analysis approach, the metamodel creation method, and the modifiability metamodel. The strengths with this kind of documentation are: 1) stability - documents can be reviewed repeatedly, 2) unobtrusiveness - documents are not created as a result of the study, 3) exactness - documents usually contain names, references, details, etc and 4) broad coverage - documents have long time span, cover many events and settings. Weakness with literature studies are: 1) the selection of documents can be biased and 2) reporting in the documentation can be biased [199, 200].

Interviews and surveys were used in the second development phase of the metamodel and during the validation. Strengths with these methods are that they: 1) can be targeted - focusing directly on the topic at hand and 2) are insightful - providing perceived causal inferences. Weaknesses with interviews and surveys are: 1) question bias due to inaccurately constructed questions, 2) response bias, 3) inaccuracies due to poor recollection and 4) reflexivity - respondents might answer what they believe the researcher wishes to hear [199, 200].
Developing the EA analysis approach
Developing the metamodel creation method
Developing the modifiability metamodel
Validating the method & metamodel

<table>
<thead>
<tr>
<th>Data collection methods</th>
<th>Developing the method for enterprise software system modifiability analysis using enterprise architecture models</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Developing the EA analysis approach</td>
</tr>
<tr>
<td>Documentation / literature study</td>
<td>✓</td>
</tr>
<tr>
<td>Surveys</td>
<td>✓</td>
</tr>
<tr>
<td>Interviews</td>
<td>✓</td>
</tr>
<tr>
<td>Archival records</td>
<td></td>
</tr>
</tbody>
</table>

Table 1.3: The use of different methods for data collection.

Archival records were used during some of the case studies in the validation phase. Strengths with archival records are: 1) stability - records can be reviewed repeatedly, 2) unobtrusiveness - records are not created as a result of the study, 3) exactness - records usually contain names, references, details, etc, 4) broad coverage - records have long time span, cover many events and settings and 5) preciseness and quantitative nature. Weakness with archival records are: 1) selection and reporting bias and 2) accessibility due to privacy reasons [199, 200].

Analysis

A desired feature of the method for modifiability analysis is the ability to assess and estimate change cost. Bayesian analysis is a well known method that can be used for that purpose. Therefore, the enterprise architecture analysis approach proposed employs Bayesian analysis. Both the extended influence diagrams (Paper A and B) and the probabilistic relational models (Paper C to E) are based on Bayesian networks.

Friedman et al. [201] describes a Bayesian network, \( B = (G, P) \), as a representation of a joint probability distribution, where \( G = (V, E) \) is a directed acyclic graph consisting of vertices, \( V \), and edges, \( E \). The vertices denote a domain of random variables \( X_1, \ldots, X_n \), also denoted chance nodes. Each chance node, \( X_i \), may take on a value \( x_i \) from the finite domain \( Val(X_i) \). The edges denote causal dependencies between the nodes, i.e. how the nodes relate to each other. The second component, \( P \), of the network \( B \), describes a conditional probability distribution (CPD) for each chance node, \( P(X_i) \), given its parents \( Pa(X_i) \) in \( G \).

Using a Bayesian network, it is possible to answer questions such as what is the probability of \( X = x_1 \) given that \( Y = y_2 \) and \( Z = z_1 \). An example of a Bayesian network and a CPD representing the chance nodes \( X \), \( Y \), and \( Z \) and how these relate is shown in Fig. 1.7. The CPD next to the network answers the question stated above. More comprehensive treatment on Bayesian networks can be found in e.g. [202, 203, 204, 205].

![Figure 1.7: A Bayesian network and a conditional probability distribution for chance node X given Y and Z.](image)
Validity and reliability

During the research work, several issues concerning the validity have been addressed. Multiple sources in both the development phase and the validation phase were used. Key stakeholders have been addressed for reviews of drafts. A chain of evidence was established. Theory was developed and used. There are however some threats to the validity and these mainly concern the 21 change projects studied in the validation phase when focusing on the estimation capabilities, cf. Paper E. Since there was no possibility of studying projects from start to end most data gathered in these case studies was collected after the projects were finished. Thus providing a final cost of the projects for comparison, but also inferring an uncertainty of the a priori value of the data. Another issue being a threat to the validity concerns the scales used when collecting and analyzing data, especially the transformation between these scales. The cost interval segmentation used when estimating the costs is also a validity issue since this so far only has been done after the projects were finished.

Reliability was achieved by documenting the research work during all phases. All case studies have their own reports describing data collection and analysis. The aim has been to operationalize as many steps as possible, e.g. conducting surveys with predefined choices and using methods with well defined guidelines and rules, thereby allowing other investigators to repeat the work.

Work progress

The research started with Paper A followed by Paper B. During this period several other publications, not included in this thesis, provided valuable input for the research. This constituted the first version of the EA method for modifiability analysis. Paper A and B, together with the 30 related publications, where then followed by the three simultaneously written papers C, D, and E. Together, these three contain the second version of the EA method for modifiability analysis. The work progress is illustrated in Fig 1.8.

![Timeline diagram showing the order of publications from 2006 to 2009.](image-url)

Figure 1.8: The order in which the included publications were written.
Bibliography


Part II

Papers A to E
Chapter 2

Paper A: Enterprise Architecture Analysis with Extended Influence Diagrams
Enterprise architecture analysis with extended influence diagrams

Pontus Johnson · Robert Lagerström · Per Närman · Mårten Simonsson

Published online: 12 May 2007
© Springer Science + Business Media, LLC 2007

Abstract The discipline of enterprise architecture advocates the use of models to support decision-making on enterprise-wide information system issues. In order to provide such support, enterprise architecture models should be amenable to analyses of various properties, as e.g. the level of enterprise information security. This paper proposes the use of a formal language to support such analysis. Such a language needs to be able to represent causal relations between, and definitions of, various concepts as well as uncertainty with respect to both concepts and relations. To support decision making properly, the language must also allow the representation of goals and decision alternatives. This paper evaluates a number of languages with respect to these requirements, and selects influence diagrams for further consideration. The influence diagrams are then extended to fully satisfy the requirements. The syntax and semantics of the extended influence diagrams are detailed in the paper, and their use is demonstrated in an example.

Keywords Enterprise architecture models · Formal language · Influence diagrams

1 Introduction

During the last decade, enterprise architecture has grown into an established approach for holistic management of the information systems in an organization. Enterprise architecture is model based, in the sense that diagrammatic descriptions of the systems and their environment constitute the core of the approach. A number of enterprise architecture initiatives have been proposed, including The Open Group Architecture Framework (TOGAF) (The Open Group 2005), the Zachman Framework (Zachman 1987), GERAM (IFAC-IFIP 1999), CIMOSA (Kosanke 1995), PERA (Williams 1994), DoDAF (Department of Defense 2004), Intelligrid (Hughes 2004) and more.

What constitutes a “good” enterprise architecture model has thus far not been clearly defined. This is due to the fact that the “goodness” of the model is not an inherent property, but contingent on the purpose the model is intended to fill, i.e. what kind of analyses it will be subjected to Johnson and Ekstedt (2007); Lindstrom et al. (2006). For instance, if one seeks to employ an enterprise architecture model for evaluating the interoperability of an information system, the information required from the model differs radically from the case when the model is used to evaluate the system’s usability.

Enterprise architecture analysis is the application of property assessment criteria on enterprise architecture models. For instance, one investigated property might be the information security of an information system and a criterion for assessment of this property might be “If the architectural model of the enterprise features an intrusion detection system, then this indicates a higher level of information security than if there is no such
system.” Criteria and properties such as these may be extracted from scientific theories, or from empirical measurements. In current practice, employed criteria are rarely expressed explicitly. If they are, they are frequently both vague and ambiguous in nature, making enterprise architecture analysis problematic with respect to both precision and accuracy.

To ameliorate such analyses, a stringent method for representing the assessment criteria would be useful. Furthermore, the analyses would benefit greatly if the considered property could be evaluated in a quantitative fashion. A notation for expressing the preference criteria in an unambiguous manner coupled with an inference mechanism for making deductions would fulfill these demands. This article proposes the extended influence diagrams as such a language. Employing the probabilistic inference mechanism of Bayesian networks, extended influence diagrams allow the analysis of a wide range of important properties of enterprise architecture models.

This paper unfolds as follows. Section 2 presents the context of enterprise architecture-based decision making. This leads to a set of requirements on potential languages for the support of such decision making, presented in Section 3. Section 4 considers to what extent a number of existing languages fulfill these requirements. Influence diagrams turn out as the most suitable candidates, so these are presented in Section 5. However, as influence diagrams do not fulfill all requirements, Section 6 proposes a set of extensions to these in order to increase the usefulness with respect to the IT decision making context. Section 7 presents the construction and use of these extended influence diagrams and in Section 8 this is demonstrated with an example. Finally, the article is concluded in Section 9.

2 Enterprise architecture analysis

The purpose of having enterprise architecture models and conducting analyses of these is to facilitate the making of rational decisions about information systems in an organization. A rational decision maker is an agent facing an alternative after a process of deliberation in which he or she answers three questions: “What is feasible?”, “What is desirable?” and “What is the best alternative according to the notion of desirability, given the feasibility constraint?” (Rubenstein 1998; Howard 1988). Translating these questions into the context of enterprise architecture analysis, architectural scenarios answer the first question of what is feasible. Answers to the second question regarding desirables, are often expressed in terms of the change of various information system properties such as increased information security, increased interoperability, increased availability, etc. The answer to the third question, providing the link between the feasible to the desirable, i.e. the link between the scenarios and the properties of interest, is given by architectural analysis (cf. Fig. 1).

Fig. 1 Enterprise architecture scenarios are subjected to analysis in order to determine their qualities (in terms of security, availability, performance, etc.)
Decision making, whether the decisions apply to IT or not, is rarely performed under conditions of complete certainty. One fundamental uncertainty is regarding the definition of various concepts. This is called *definitional uncertainty*. For instance, some authors define the term information security in terms of confidentiality, integrity and availability, (Liu et al. 2000), whereas others add the concepts of non-repudiation and accounting to the definition (Poslad and Calisti 2000).

Related to definitional uncertainty is *theoretical heterogeneity*. Existing knowledge regarding the nature of enterprise information systems is not consolidated within a single commonly accepted framework, as may be the case for more mature disciplines. A consequence of this is that there is a need to relate similar concepts to each other, and to relate concepts at varying levels of abstraction to each other.

In addition to definitional uncertainty, there may also be uncertainty with regard to how the world actually behaves. Knowledge of exactly how various phenomena affect one another is seldom certain. There is a level of *causal uncertainty*. An example of this would be uncertainty with respect to the causal effect of the percentage of systems with updated virus protection may have on the level of information security.

When creating architecture models of an enterprise, the information represented in the models is normally associated with a degree of uncertainty. Perhaps the information was collected a while ago and has now become obsolete, or perhaps the information was gathered from a source that might have been incorrect. In these situations, the decision maker suffers from *empirical uncertainty*.

From this description of the context of enterprise architecture analysis, requirements may be extracted on the languages used for analysis specification. Such requirements are presented in the next section.

### 3 Requirements on language

The previous section presented the context for which a language for enterprise architecture analysis is sought. Summarizing, this context is a decision-making situation characterized by three kinds of uncertainty: definitional uncertainty, causal uncertainty, and empirical uncertainty. Also, the language needs to have facilities for managing theoretical heterogeneity.

A language appropriate for this context needs to be able to represent a number of things. Firstly, basic decision-making support requires that the notation is able to represent the decision maker's goals, domain of control, as well as the causal relations between that which can be controlled and the desired goals. As an example, the goals in a decision situation related to information systems, could be to increase system performance. The decision alternatives of that same decision situation could be either to buy new hardware with faster processors, or to implement a new, more streamlined software architecture.

Secondly, in order to manage definitional uncertainty, the language needs to be able to clarify what is meant by the represented concepts, it needs to be able to define vague concepts. Readers of language specifications should be able to understand precisely what is meant by terms such as for instance “system performance”.

Thirdly, the notation needs to be able to express causal and empirical uncertainty, i.e. uncertainty with respect to how concepts affect each other and with respect to the measured values of the concepts.

Finally, in order to manage heterogeneous theory, the language needs to allow multiple levels of abstraction. The language must be able to capture that some concepts and phenomena are on a different level of abstraction than other. It should for instance be possible to express that the concept “performance” is more abstract than the concept “throughput”.

### 4 Evaluation of existing languages

A number of languages have already been proposed for various decision-making contexts. In this section, we will consider a number of those languages with respect to the requirements presented in the previous section.

For the modeling and analysis of uncertain phenomena, Bayesian networks are often proposed (Neapolitan 2004). Bayesian networks are composed of nodes with associated values, and arcs between the nodes. The nodes' probabilistic dependencies on each other are specified by the arcs and by so called conditional probability distributions. With respect to our requirements, Bayesian networks provide support for modeling causality, causal uncertainty and empirical uncertainty. There is, however, no support for goals, controllable variables, definitional relations, or multiple abstraction levels.

Bayesian networks have been extended with utility and decision nodes in order to better support rational decision making, i.e. to represent goals and domains of control. Diagrams that feature such nodes are called influence diagrams (Shachter 1986, 1988). In order to allow the representation of multiple levels of abstraction, object-oriented influence diagrams have been proposed.
Fig. 2 Degree of requirement fulfillment by existing languages. The Y-axis consists of the different requirements as specified in Section 3 and the X-axis contains the various languages as described in the present chapter. Dotted check-notation denotes partial fulfillment.

(Hugin Expert A/S 2004). These allow concepts to be aggregated into objects, which can be expanded and collapsed. As will be explained in Section 6, however, object-oriented influence diagrams do not offer support for avoiding definitional uncertainty.

An alternative to the abovementioned Bayesian approaches is Dempster–Shafer theory (Shafer 1976; Yang 2001). Dempster–Shafer is specifically designed to allow the representation of ignorance, something that the Bayesian formalisms only are capable of to a limited degree. Dempster–Shafer Theory, however, lacks support for decision and goal representation, as well as support for managing multiple levels of abstraction or defining new concepts.

I* (or I-star) is a goal-oriented description framework depicting actors, their goals and dependencies between the actors and the goals (Yu 1996). Designed primarily to facilitate business process management and requirements engineering, I* is able to represent goals, causal relations between concepts, and via it’s task break-down approach I* is able to manage different levels of abstraction. I* is however unable to deal formally with the many different uncertainties of the world and offers no support for defining vague concepts in a stringent manner. Moreover, I* does not represent the domain of control.

The perhaps best known modeling notation in the information systems area is the Unified Modeling Language (UML) (Fowler and Scott 1997). The class diagrams in this language explicitly supports the specification of definitional relations and multiple abstraction levels, but no other of the requirements presented in Section 3.

Figure 2 summarizes the respective notations’ degree of fulfillment with respect to the considered requirements. As the figure shows, influence diagrams and object oriented influence diagrams are the most suitable candidates. These are therefore presented in greater detail in the next section.

5 Influence diagrams

This section briefly describes conventional influence diagrams. For a more comprehensive treatment, the reader is referred to Shachter (1986, 1988); Howard and Matheson (2005); Jensen (2001) and Neapolitan (2004).

An influence diagram is an extension of Bayesian networks. As was mentioned above, a Bayesian network graphically represents causal relations between nodes, where each node represents a variable with a number of states. Moreover, Bayesian networks are able to represent the uncertainty of the causal relations using probabilistic reasoning. Using the terminology from Section 2 above, a Bayesian network is able to answer the question “what is feasible?”, through the modelling of the real world. An example of a Bayesian network is shown in Fig. 3. The example shows that there is a causal dependence between the variable “time spent studying mathematics”, and the variable “understanding of mathematics” and the direction the arrow points suggests that the former affects the latter, rather than vice versa.

To capture that most relations are probabilistic rather than deterministic in nature—in some cases a student could spend a considerable time studying and still not get a better understanding of the subject—Bayesian networks use so called conditional probability matrices. A conditional probability matrix captures the likelihood of a variable being in a state X, under the condition that it’s “predecessor” is in a state Y. See Fig. 4.

In addition to representing causal relations, influence diagrams support a more complete and intuitive...
description of decision problems, stating both what is desired, and what alternatives are available. This is accomplished by the introduction of two new nodes. The first is the decision node, which represents the decision alternatives at hand. The second is the utility node where the outcome of a decision is quantitatively assessed as an expected utility. To illustrate, we expand the example in Fig. 3 to include also the decision node “Decision to study” wherein the student decides whether to study mathematics or not. Also included is the utility node “Mathematics grade” which in this particular case is the variable that the student wishes to maximize. In this simple example, the decision analysis is straightforward and suggests that the student ought to decide to study. See Fig. 5.

More rigorously, an influence diagram is a network used for modelling uncertain variables and decisions, consisting of a directed graph \( G = (N, A) \). There are three types of nodes in the set \( N \), partitioned into the sets \( V, C \) and \( D \). There is at most one utility node \( v \in V \), depicted as a rhombus (Fig. 6). There are zero or more chance nodes \( c \in C \), these are depicted as ovals. There may be zero or more decision nodes \( d \in D \), and these are depicted as squares.

There are two types of arcs in the set \( A \), partitioned into the sets \( K \) and \( I \). Arcs into utility and chance nodes are causal, \( k \in K \), representing probabilistic dependence. Arcs into decision nodes are informational, \( i \in I \), and imply time precedence. Both the causal relation arcs and the informational relation arcs are depicted as arrows. See Fig. 6.

The set of causal predecessors are represented by \( CP(i) = \{ j \in N : (i, j) \in K \} \), where \( i \in \{ V, C \} \). The set of informational predecessors are represented by \( IP(i) = \{ j \in N : (i, j) \in I \} \), where \( i \in D \).

Associated with each node \( i \) is a variable \( X_i \) and a set \( \Omega_i \) of possible values it may assume. If \( i \) is the utility node, then \( X_i \) represents the expected utility and its domain \( \Omega_i \) is a subset of the real line. If \( i \) is a chance node, then \( \Omega_i \) is the sample space for the random variable \( X_i \). Finally, decision node \( i \) has alternative \( X_i \) chosen from the set \( i \). The utility node \( v \in V \) has an associated utility function \( U : \Omega_{CP(v)} \to \Omega_v \), which represents the expected utility as a function of the values of the conditioning predecessors of the utility node. There is a conditional probability distribution, \( Pr \), for every chance node \( i \), given the values of its causal predecessors, \( Pr[x_i|x_{CP(i)}] \). The notation \( Pr[x_i|x_{CP(i)}] \) means that \( Pr(X_i = x_i|X_{CP(i)} = x_{CP(i)}) \).

The probability distributions for each chance node are represented in conditional probability matrices. Figure 7 shows the conditional probability matrix for a chance node \( y \) dependent on a node \( z \).

Object-oriented influence diagrams (Hugin Expert A/S 2004) follow the same rules that influence diagrams do, except that they also provide support for composition of sub-diagrams. The main diagram may thus consist of a set of sub-diagrams, all consisting in turn of nodes or sub-sub-diagrams. Nodes may be related between sub-diagrams.

### 6 Extended influence diagrams

Although influence diagrams may be used for enterprise architecture analysis in their conventional form, there are, as mentioned in Section 4, some requirements that are not sufficiently addressed. This section therefore presents a set of extensions to influence diagrams for the purposes of architecture analysis.

<table>
<thead>
<tr>
<th>Time spent studying</th>
<th>Good</th>
<th>5 hours</th>
<th>3 hours</th>
<th>1 hour</th>
<th>0 hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Understanding of Mathematics</td>
<td>0.8</td>
<td>0.5</td>
<td>0.2</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>Passable</td>
<td>0.15</td>
<td>0.4</td>
<td>0.2</td>
<td>0.15</td>
<td></td>
</tr>
<tr>
<td>None whatsoever</td>
<td>0.05</td>
<td>0.1</td>
<td>0.6</td>
<td>0.8</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 4 A conditional probability matrix showing the probability of a student’s understanding of mathematics given the time spent studying. In this case it is very probable (80%) that a student will have a good understanding if he or she spends 5 h studying, but it is not certain.

Fig. 5 A very simple influence diagram showing not only the causal dependence between time spent studying mathematics and understanding mathematics, but also detailing the student’s goals and alternatives.
6.1 Lexically defined nodes

Causality means that one phenomenon in the real world somehow affects another. Because causality is a concept of the real world, it is important that there is a mapping between the real world and the influence diagrams, i.e., that the concepts presented in the influence diagrams are well-defined. If they are not well-defined, it will be impossible to determine whether there is in fact any truth to the causal relation between the phenomena.

The definitions of some nodes in influence diagrams are deemed uncontroversial. For instance, in the IT community, we might assume that there is common agreement on the definition of the concept of memory size; it is typically measured in terms of bytes. Nodes that we consider uncontroversially defined are called *lexically defined nodes*, $L \subset N$ (see Fig. 8) (Scriven 1954).

6.2 Stipulatively defined and undefined nodes

In the considered decision-making context, we have assumed that there is considerable confusion as to the meaning of many concepts, such as information security. There are thus many potential nodes that are not lexically defined. In order to manage these, we introduce the possibility to define nodes within the influence diagram. This is done by relating a node, directly or indirectly, to lexically defined nodes with a new kind of arc, called a definitional relation, $\delta \in \Delta$, where $\Delta \subset A$ (see Fig. 8). Nodes that are defined in the diagram using the definitional relation are called *stipulatively defined nodes*, $S \subset N$ (Scriven 1954). Those nodes that are neither lexically nor stipulatively defined are simply *undefined*, $U \subset N$.

6.3 Definitional relations

This subsection details the semantics of definitional relations, $\Delta$. The set of defining nodes is given by $DF(i) = \{ j \in N : (j, i) \in \Delta \}$ where $i \in \{V, C\}$. The definitional relation implies simple aggregation; the defined node is comprised of its constituent parts. The definitional relation is represented mathematically by the same conditional probability distributions as are used for causal relations, $Pr(x_i|DF(i))$. Since definitional relations are simple aggregations, a node that has defining nodes cannot also in the same graph feature causal predecessors.

The concept of definitional relations is similar to the aggregation mechanism provided by object-oriented influence diagrams. However, aggregated objects in object-oriented influence diagrams are to be viewed more as placeholders for diagrams than as nodes. They are therefore not associated with any variable $X_i$. As is considered below, the association of a variable with the aggregated node is of significant importance for our purposes.

6.4 Controllability of nodes

A utility node may be affected by many chance nodes, and it may be the case that only some of these chance nodes are affected (directly or indirectly) by the decision node. Those nodes that lie in a causal path from decision node to utility node or are stipulatively defined by such nodes are relevant for decision making.
Other nodes are not. Nodes that are completely dependent on the decision node are denoted *controllable nodes*, $F \subset N$. Nodes that are completely independent of the decision nodes are denoted *uncontrollable nodes*, $W \subset N$. Finally, nodes that are partially dependent on the decision node are denoted *semi-controllable nodes*, $SC \subset N$. The graphical rendering of these node types is given in Fig. 8.

Furthermore, it may sometimes be useful to distinguish between *directly controllable nodes*, $DC \subset F$ and *indirectly controllable nodes*, $IDC \subset F$. Directly controllable nodes are targets of a causal relation originating in a decision node, $DC(i) = \{ j \in D : (j, i) \in K \}$. Indirectly controllable nodes are targets of a causal relation originating in a directly controllable node or an indirectly controllable node, $IDC(i) = \{ j \in \{ DC, IDC \} : (j, i) \in K, \Delta \}$.

### 6.5 A set of related influence diagrams

One major benefit of definitional relations is that it is possible to treat a topic on different levels of abstraction. It is thus possible to speak abstractly of the causal effect that for instance firewall protection has on information security, and more concretely about the effect it has on confidentiality, integrity and availability. These two levels of abstraction are presented in Fig. 9 in terms of two different graphs.

We define an extended influence diagram as a set of graphs, $E = \{ G^1, ..., G^n \}$, related by expansions and collapses of the definitional structures, where the expansions and collapses are dictated by specific transforms, $G^i = \text{coll}(G^j)$ and $G^i = \text{exp}(G^j)$. In Fig. 9, $G^2 = \text{coll}(G^1)$ and $G^1 = \text{exp}(G^2)$. The transforms are required to ensure that the graphs do not contradict each
other. In the rest of this subsection, these transforms are detailed.

Firstly, the effects of node collapses on the affected conditional probability distributions are considered. In Fig. 10, given a graph \( G^1 \) with an expanded node \( i \), we need to know how to represent the conditional probability distributions of the nodes affected by the collapse, namely node \( i \) and nodes \( k_1, \ldots, k_q \) in \( G^2 \).

Using the law of total probability we find that the conditional probability distribution for node \( i \) in the initial expanded diagram is

\[
Pr(x_i|x_j, \ldots, x_{j_p}) = \sum_{x_{a1}, \ldots, x_{am}} Pr(x_i|x_{a1}, \ldots, x_{am}) \\
\times \prod_{x_a = a1, \ldots, am} Pr(x_a|x_j, \ldots, x_{j_p})
\]

The notation \( \sum_{x_{a1}, \ldots, x_{am}} \) means the sum as the variables \( x_1, \ldots, x_m \) go through all possible values in their corresponding spaces. For node \( \kappa \in \{k_1, \ldots, k_q\} \), we have

\[
Pr(x_\kappa|x_i) = \sum_{x_{a1}, \ldots, x_{am}} Pr(x_\kappa|x_{a1}, \ldots, x_{am}) \\
\times \prod_{x_a = a1, \ldots, am} Pr(x_a|x_i)
\] (1)

The first term on the right hand side of Eq. 1 is given. The second term can be calculated with Bayes' theorem, \( P(A|B) = P(B|A) P(A) / P(B) \),

\[
Pr(x_\kappa|x_i) = \frac{Pr(x_i|x_\kappa) Pr(x_\kappa)}{Pr(x_i)}
\] (2)

All three terms in this equation need to be expanded; the law of total probability can be used for this purpose.

\[
Pr(x_i|x_a) = \sum_{x_{a1}, \ldots, x_{am}} \prod_{x_a = a1, \ldots, am} Pr(x_a|x_{1}^n) \\
Pr(x_a) = \sum_{x_j, \ldots, x_{j_p}} \prod_{\eta = j1, \ldots, j_p} Pr(x_\eta)
\]

\[
Pr(x_\kappa) = \sum_{x_a = a1, \ldots, am} \prod_{\kappa \in \{k_1, \ldots, k_q\}} Pr(x_\kappa)
\]

For nodes \( \eta \in \{j_1, \ldots, j_p\} \) are provided as prior probabilities. The most expanded graph is considered as the “true” graph, with which the more collapsed versions need to comply according to the node collapse transformations above. In the case where there is already an expanded graph in the graph set \( E \), a node expansion is simply a return to that graph. During diagram construction, it may however also be convenient to have access to a node expansion transformation, \( G^1 = exp(G^2) \), cf. Fig. 10. In this case, we assume that \( Pr(x_i|x_{a1}, \ldots, x_{am}) \) is given. We then need to know how to represent the conditional probability distributions of the nodes affected by the expansion, namely nodes \( a_1, \ldots, a_m \) and nodes \( k_1, \ldots, k_q \). For \( \alpha \in \{a_1, \ldots, a_m\} \),

\[
Pr(x_\alpha|x_j, \ldots, x_{j_p}) = \sum_{x_i \in \Omega_i} Pr(x_\alpha|x_i) Pr(x_i|x_j, \ldots, x_{j_p})
\]

\( Pr(x_\kappa|x_j, \ldots, x_{j_p}) \) is given, and \( Pr(x_\alpha|x_i) \) was calculated in Eq. 2. For \( \kappa \in \{k_1, \ldots, k_q\} \),

\[
Pr(x_\kappa|x_{a1}, \ldots, x_{am}) = \sum_{x_i \in \Omega_i} Pr(x_\kappa|x_i) Pr(x_i|x_{a1}, \ldots, x_{am})
\]
At this point, let us summarize the mechanisms of node collapse and expansion. If we have \( \Pr\{x_{kp}, x_{a1}, ..., x_{am}\} \), \( \Pr\{x_k|x_{a1}, ..., x_{am}\} \), \( \Pr\{x_a|x_j, ..., x_{jp}\} \), and \( \Pr\{x_\eta, \eta \in \{j_1, ..., j_p\}\} \) for graph \( G_1 \), we can calculate \( \Pr\{x_\alpha|x_j, ..., x_{jp}\} \) for graph \( G_2 \). And vice versa (in the case where \( G_1 \) does not already exist), given \( \Pr\{x_\kappa|x_i\} \), \( \Pr\{x_i|x_j, ..., x_{jp}\} \), and \( \Pr\{x_\eta\} \) for graph \( G_2 \) we can calculate \( \Pr\{x_k|x_{a1}, ..., x_{am}\} \), \( \Pr\{x_a|x_{a1}, ..., x_{am}\} \), and \( \Pr\{x_a|x_j, ..., x_{jp}\} \) for graph \( G_1 \).

### 7 Extended influence diagrams for enterprise architecture analysis

Recalling the decision making approach described in Section 2, the question “What is desirable?” may be specified by an extended influence diagram utility node, \( v \in V \) (see Fig. 11). An example of an utility node concept for enterprise architecture analysis is Information Security.

The answer to “What is feasible?” is in the case of architecture-based decision making answered by a set of enterprise architecture scenarios, \( \Theta = \{\theta_1, ..., \theta_n\} \). The characteristics of such architectural scenarios may then be used to determine the conditional probability distributions of chance nodes in the extended influence diagram, \( \Pr\{x_i\} = f(\theta_i) \). The decision maker’s choice between architecture scenarios is represented by a decision node, \( \theta \in \Theta \), which is connected by causal relations to all those chance nodes that the scenario selection might affect.

The chance nodes that are directly affected by the decision node also need to be related to the utility node so that the utility of the different scenarios can be calculated. Exactly how these links are specified is an important issue and will be addressed later in this paper. Typically, it is performed by the use of intermediary chance nodes. The resulting influence diagram is a representation of how we believe that the real world functions; it is the representation of a theory of, for instance, information security. The diagram thereby provides the answer to the decision maker’s third question, “What is the best alternative according to the notion of desirability, given the feasibility constraint?”

The rest of this chapter describes a generic process for construction of extended influence diagrams and the use of them for enterprise architecture analysis. The construction process is described in Section 7.1 while the process of enterprise architecture analysis is described in Section 7.2.

#### 7.1 Development of extended influence diagrams

Figure 12 depicts a process for the construction of extended influence diagrams. This subsection describes that process step by step. The focus of this article is on the conceptual rather than practical aspects of the...
7.1.1 Step one: Introduce decision node

At the very start, the decision node should be identified. As mentioned above, in enterprise architecture analysis, decision nodes represent the choice between different enterprise architecture scenarios, such as the choice between integrating a set of systems, replacing them, or maintaining the status quo.

7.1.2 Step two: Introduce utility node

The second step is to define the utility node. The utility node is the target of the enterprise architecture analysis. Examples of utility nodes are information security, modifiability, performance and reliability. When defining the utility node, its variable type should also be decided on, e.g. \{Low, Medium, High\}, \{Present, Absent\}, \{True, False\}, or \{0, 1, 2, 3, 4\}.

7.1.3 Step three: Is the node defined?

The third step is to determine whether the node is lexically defined, stipulatively defined or undefined. Recall that a node is lexically defined if we can assume that there is common agreement on its definition. Although almost all definitions can be challenged, concepts that would normally qualify are lexically defined include weight in kilograms, number of processors or number of users. For many concepts, however, such as information security, architecture quality and competence there is no universal definition even by very pragmatic standards.

If the node is not lexically defined, it needs to be stipulatively defined. This is accomplished with the definitional relationship presented in the previous chapter. As an example, the node Availability might be defined by the two nodes Mean Time To Failure and Mean Time To Repair. The stipulative definition of a node results in the introduction of new nodes into the diagram. When new nodes are introduced, this affects the conditional probability matrices of the child nodes. These must therefore also be specified, thereby detailing the dependencies between the parents and the child. There are many approaches for efficient specification of conditional probabilities that may be employed (Druzdzel and van der Gaag 2000). When an undefined node has been stipulatively defined, the diagram construction process refocuses on one of the new nodes and returns to the third step.

7.1.4 Step four: Is the node uncontrollable?

For each defined node, the fourth step considers whether the node is uncontrollable. Recall that uncontrollable nodes are unaffected by variable changes in the decision node; this means that potential variations in the value of the uncontrollable node are unrelated to the choices of the decision maker. If the node is uncontrollable, it will not provide decision supporting information, so there is no need to continue exploring this branch of the diagram. The node is therefore deleted, the process refocuses on the next unexamined node and returns to the third step.

7.1.5 Step five: Is the node semi-controllable?

Step five queries whether the node under consideration is semi-controllable. Semi-controllable nodes are affected both by the decision makers choices and other uncontrollable, phenomena. If a node is semi-controllable, it is important to separate those aspects which are controllable from those which are not. Therefore, new nodes are introduced into the diagram with causal relations to the semi-controllable node under consideration. As an example, we might believe that the semi-controllable node Mean Time To Repair is causally affected by both the Maintainability of the system and Flexibility of Working Hour Regulations. Of these, the maintainability might be controllable, while the working hour regulations might be beyond the decision makers domain of control. In the same manner as in the second step, the involved conditional probability matrices need to be specified. The diagram construction process once again refocuses on one of the new nodes and returns to the third step.

7.1.6 Step six: Is the node directly controllable?

In the sixth step, the process considers whether the nodes are directly or indirectly controllable. A directly controllable phenomenon is an immediate consequence of the decision makers choice. For instance, if a scenario is chosen where one system is replaced by another, this may directly entail that the CPU speed is increased. Directly controllable nodes are causally connected to the decision node. For nodes that do not seem to be directly controllable, one or several new nodes are introduced into the diagram, the focus shifts to the first of these, and the process returns to the third step.

When the whole process is finished, the result is an extended influence diagram where the nodes are causally affected by the decision node and in turn either
7.2 Analyzing with extended influence diagrams

When an extended influence diagram has been constructed, it may be used to compare and assess the quality of different enterprise architecture scenarios.

7.2.1 The enterprise architecture metamodel

If the extended influence diagram is regarded as the algorithm, then the enterprise architecture model may be viewed as the data upon which the algorithm operates. The result of executing the algorithm on the data is the value of the utility node, i.e. how “good” the model is with respect to the assessed concept (information security, availability, etc.). In the same way that it is important that the data fits the algorithm in the general case, it is here necessary that the enterprise architecture model fits the extended influence diagram. More precisely, it is required that the enterprise architecture model contains at least the information required by the extended influence diagram. Whether this is the case...
can easily be controlled by comparing the extended influence diagram to the relevant enterprise architecture metamodel (Johnson et al. 2004).

7.2.2 Linking enterprise architecture scenarios to extended influence diagrams

In extended influence diagrams, decision nodes represent a choice between alternatives. In enterprise architecture analysis, these alternatives are concretized by different enterprise architecture scenarios. We will now consider how the information represented in the enterprise architecture scenarios is introduced into the extended influence diagrams. Figure 13 schematically depicts this link.

In the figure, the enterprise architecture model named Scenario X contains a set of entities. Considering one of these entities, say the System A entity, we find that it features a set of attributes, Memory Size, Lines Of Code, etc. As is demonstrated in the figure, the value of each of these attributes is represented in a conditional probability matrix, \( Pr\{\text{Memory Size}_{EA} = \text{msi}\} \). Note that this probabilistic representation of attributes is not standard in enterprise architecture modeling; however, the approach retains the possibility to present attribute values deterministically by allowing only zero or unity probabilities in the matrix.

Also present in the figure is the extended influence diagram. The directly controllable nodes, i.e. the chance nodes that are directly linked to the decision node, constitute the coupling to the enterprise architecture model. Examining one of these nodes in detail, we find that it is named Memory Size. The node’s conditional probability matrix is presented in the figure. The link between the enterprise architecture model and the extended influence diagram is concretely specified by the following requirement: the Memory Size node’s probability given that Scenario X is selected is equal to the conditional probability of the Memory Size attribute of the System A entity, i.e. \( Pr\{\text{Memory Size}_{EID} = \text{msi}\} = Pr\{\text{Memory Size}_{EA} = \text{msi}\} \).

7.2.3 Calculating the results

The conditional probability distributions of the directly controllable nodes are thus retrieved from the enterprise architecture scenarios. The higher-level conditional probability matrices were determined already during the extended influence diagram construction process. The value of the utility node can therefore be calculated employing standard methods (Jensen 2001; Neapolitan 2004). There are several tools available on the market for these calculations, such as Genie (2006) and Hugin (2004).

8 An example enterprise architecture analysis

This section provides an example of the construction of extended influence diagrams and the use of them for enterprise architecture analysis. The former is described in Section 8.1 and aims at demonstrating not only the development process in itself, but also the expressive possibilities of the extended influence diagram language. The perspective of the diagram user is described in Section 8.2, where the developed extended influence diagram is used by a decision maker. The context is that of Section 2, so the decision maker in question is an IT decision maker. The example relates to information security and to the ISO 17799 standard (International Organization for Standardization 2000).
8.1 Development of extended influence diagrams

A process for the development of extended influence diagrams described in the previous section and in Fig. 12 is followed in the presented example.

8.1.1 The decision and the utility node

We imagine a decision situation in an enterprise where the IT decision maker faces the choice between two different future scenarios, Scenario 1 and Scenario 2. We start with the assumption that the general goal is to maximize the company profit. The decision node Scenario Selection thus represents the decision, while the utility node Profit represents the goal, see Fig. 14.

Following the flowchart of Fig. 12, we assume that the term profit is lexically defined; for the purpose of this example the ambiguity inherent in the concept will not cause any misunderstandings. The flowchart thus directs us to the subsequent questions, regarding controllability.

We posit that Profit is not uncontrollable; the actions taken by IT decision makers at a company causally affects the profits of the company. Moving on in the flowchart, we find that the concept is, however, semi-controllable in the sense that nodes not controlled by
the IT decision maker have some level of impact on the utility node.

8.1.2 Causally related nodes

It is not satisfactory to use a semi-controllable variable as a basis for decision making, since its value may be affected by phenomena that are unrelated to the decision situation at hand. The flowchart therefore suggests the identification and incorporation of new chance nodes with a causal impact on Profit. For the purpose of this example, the chance nodes Competitive Advantage, Efficiency and Company Culture are introduced. See Fig. 15. Although not presented here, the utility function, \( U \), detailing the relation between the Profit node and its causal predecessors must also be specified.

Restarting at the top of the flowchart for the new nodes, none of these concepts would normally be considered lexically defined; such nodes need to be defined stipulatively. However, to avoid an overly complicated example, we will assume that these nodes are lexically defined.

Still following the flowchart, we posit that the chance node Company Culture is uncontrollable; an IT decision maker is typically not in a position to significantly influence or change company culture. Therefore, this node is deleted and no more nodes will be explored below this one.

The chance nodes Competitive Advantage and Efficiency are examples of chance nodes that are semi-controllable, i.e., just as for the node Profit, the decision maker has control over some predecessors that causally influence the behavior of these chance nodes, but not all of them.

8.1.3 Definitionally related nodes

A chance node that does not suffer from semi-controllability, yet has causal effects on Competitive Advantage and Efficiency, is Information Security. This node and the resulting conditional probability distributions for Competitive Advantage and Efficiency are thus introduced. In a more complete example, one would imagine many more nodes similar to Information Security, such as Performance, Interoperability, etc.

Information security is not lexically defined, so we stipulatively define it in terms of Confidentiality, Integrity and Availability (see Fig. 16). We will, for the sake of the example, consider these new nodes as lexically defined. Availability might be measured in terms of annual uptime. Integrity might be measured as percentage of business-critical information destroyed or altered in an unauthorized manner. Confidentiality might be measured as percentage of business-critical information disclosed to unauthorized persons or processes.

Figure 17 presents the conditional probability matrix of the Information Security node. Information security is measured on a five grade scale while the three defining nodes are measured on a three graded scale.

With an eye on the flow chart, we believe that the three concepts are not only defined, but also (indirectly) controllable; the IT decision maker has consid-

---

**Fig. 20** Conditional probability matrices of Confidentiality, Integrity and Availability

<table>
<thead>
<tr>
<th>User Training Process</th>
<th>Present</th>
<th>Absent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cryptographic Control Application</td>
<td>Present</td>
<td>Absent</td>
</tr>
<tr>
<td>Confidentiality</td>
<td>High</td>
<td>0.95</td>
</tr>
<tr>
<td>Medium</td>
<td>0.05</td>
<td>0.75</td>
</tr>
<tr>
<td>Low</td>
<td>0</td>
<td>0.25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Anti-Virus Application</th>
<th>Present</th>
<th>Absent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intrusion Detection Application</td>
<td>Present</td>
<td>Absent</td>
</tr>
<tr>
<td>Integrity</td>
<td>High</td>
<td>0.95</td>
</tr>
<tr>
<td>Medium</td>
<td>0.05</td>
<td>0.95</td>
</tr>
<tr>
<td>Low</td>
<td>0</td>
<td>0.025</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Back-Up Process</th>
<th>Present</th>
<th>Absent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incident Management Process</td>
<td>Present</td>
<td>Absent</td>
</tr>
<tr>
<td>Availability</td>
<td>High</td>
<td>0.95</td>
</tr>
<tr>
<td>Medium</td>
<td>0.05</td>
<td>0.95</td>
</tr>
<tr>
<td>Low</td>
<td>0</td>
<td>0.025</td>
</tr>
</tbody>
</table>
erable influence on their values. We have thus finally found concepts that are both defined and controllable. However, we do not think that any one of these three concepts are direct controllable, which is the stopping criterion of the flowchart.

8.1.4 An abstract extended influence diagram

One commonly proposed positive causal factor with respect to the information security in an enterprise is the company’s ISO 17799 compliance, i.e. the degree to which the company follows the rules specified by the international standard ISO/IEC 17799 “Information Technology - Security Techniques - Code of Practice for Information Security Management” (International Organization for Standardization 2000). However, it may be difficult to find detailed information on exactly how the constituent parts of Information Security, i.e. Confidentiality, Integrity, and Availability, are affected by ISO 17799 Compliance. In order to represent such abstract knowledge, the node Information Security can be collapsed, as represented in Fig. 18. Continuing, assuming that the node ISO 17799 Compliance is both lexically defined and directly controllable, the flowchart prescribes the connection of the decision node to the chance nodes. A first, abstract version of the extended influence diagram is thus complete, cf. Fig. 18.

8.1.5 Refinement by node expansion

The assumption that ISO 17799 Compliance is lexically defined may be criticized, so we instead choose to refine this concept with a stipulative definition. Compliance can be measured by the degree to which the company fulfills requirements stated in individual chapters of the standard. In Fig. 19 (which by no means aims for a complete description of the standard), ISO 17799 compliance is defined by four branches: The first is Human Resources Security which in turn is defined by the existence of a User Training Process to improve system user’s security awareness. The second is Communications and Operations Management, which is definitionally linked to both the existence of a Back-up Process and Anti-Virus Applications. The third is Information Systems Acquisition, Development and Maintenance, definitionally linked to the existence of Cryptographic

---

Fig. 21 The example enterprise architecture meta-model

Fig. 22 The two enterprise architecture scenarios
Control Applications and Intrusion Detection Applications. The fourth is Information Security Incident Management, defined by the existence of a formal Incident Management Process. As discussed in Section 6, the abstract causal effect of ISO 17799 Compliance on Information Security may now, after node expansion, be refined in terms of these more concrete defining nodes’ causal effects on the Confidentiality, Integrity, and Availability.

The conditional probability matrices of Confidentiality, Integrity, and Availability are presented in Fig. 20.

Direct controllability and lexical definition is now attributed to the bottom nodes in the ISO 17799 Compliance branches. The refined version of the extended influence diagram is presented in Fig. 19.

8.2 Enterprise architecture analysis with extended influence diagrams

In this section, the extended influence diagram is combined with enterprise architecture scenario models for the purposes of enterprise architecture analysis.

Typically, the connection between corporate profit and more tangible concepts, such as information security, is a product of a corporate and IT strategic planning process. In order to separate concerns, it is common to address the more concrete aspects - information security, modifiability, performance, etc. - individually. This example is accordingly limited to the analysis of information security. Information Security is therefore viewed as the utility node.

8.2.1 The enterprise architecture metamodel

We now introduce an enterprise architecture metamodel, describing the syntax of the enterprise architecture scenario models (Fig. 21). The metamodel is limited to four abstract entities, Process, Application, Role, and Document. A set of concrete subtypes of each of these entities is further defined in the metamodel. Each metamodel entity is associated with a probability distribution representing the likelihood that the entity will in fact be implemented if the scenario is selected.

8.2.2 Enterprise architecture scenarios

In this example, the decision maker is faced with the choice between two architectural scenarios, represented in Fig. 22. Note that the scenario models comply with the aforementioned metamodel. The scenarios are similar insofar as they both include anti-virus and intrusion detection applications in addition to the company’s incident management process. The scenarios differ as follows. In the first scenario, a user training process is introduced to increase the overall user security awareness. In the second scenario, cryptographic control applications are implemented to increase the confidentiality of the systems and a back-up process is introduced to improve the availability. Furthermore, the scenarios differ in which roles and documents are present.
8.2.3 Conditional probability distributions of directly controllable nodes

If the metamodel is constructed to support the analysis, information from the scenarios can be used to determine the conditional probability matrices for the directly controllable nodes of the extended influence diagram. Figure 23 presents the conditional probability matrices of the directly controllable chance nodes. As described in Subsection 7.2.2, these matrices are derived from the scenarios.

8.2.4 Calculating the results

Given the above diagram and conditional probability matrices, the expectation value and standard deviation of the Information Security node can be calculated for the two decision alternatives. The results are presented in Fig. 24. The diagram shows the level of information security as well as the uncertainty associated with the assessment. The diagram provides support for the decision maker in the choice between three alternatives: (a) choose scenario 1, (b) choose scenario 2, or (c) increase certainty of results in order to provide better decision support for the selection between the scenarios.

In order to increase the certainty of the results, we would need to engage either in more data collection, i.e. increasing the precision of the enterprise architecture model, or in more theory development, i.e. increasing the precision of the extended influence diagram.

9 Conclusions

This article has proposed the use of a formal language to support the analysis of enterprise architectures. A set of requirements for such a language were presented and a number of candidate languages were considered with respect to these requirements. The most suitable candidate, the influence diagram notation, was extended to fully support the requirements. The resulting extended influence diagrams differ from the conventional ones in their ability to cope with definitional uncertainty, i.e. the uncertainty associated with the use of language and in their ability to represent multiple levels of abstraction. The syntax and semantics of extended influence diagrams were presented, a method for their construction and use was described and subsequently demonstrated in an example. The proposed language supports quantitative analysis of important properties of enterprise architecture models, such as information security, performance, availability, and interoperability. The language thus provides both an assessment of the utility of the architecture and the uncertainty associated with that assessment.

References


Pontus Johnson is Associate Professor at the Department of Industrial Information and Control Systems at the Royal Institute of Technology (KTH) in Stockholm, Sweden. He is the research leader of a group of fifteen researchers and PhD students focusing particularly on the analysis of enterprise architectures. In the frame of the department, Pontus also supervises a number of PhD students. He is the author of a number of journal articles, international conference articles and books. He is also active as a consultant in the area of enterprise architecture. Pontus received his MSc from the Lund Institute of Technology in 1997 and his PhD from the Royal Institute of Technology in 2002.

Robert Lagerström holds a MSc in Computer Science from the Royal Institute of Technology (KTH). Robert is currently a PhD Student at the Department of Industrial Information and Control Systems at KTH in Stockholm, Sweden. His research focus is on enterprise architectures (EA) and decision analysis of maintainability using EA-models. Robert is also a member of the Swedish Chapter committee of INCOSE (International Council on Systems Engineering).

Per Närman holds a MSc E.E. from the Royal Institute of Technology (KTH). He is currently a PhD Student at the Department of Industrial Information and Control Systems at KTH in Stockholm, Sweden. Per conducts research within the field of enterprise architecture analysis and his research will result in a system quality analysis framework, an enterprise architecture metamodel connected with said analysis framework and a methodology that supports resource-efficient modeling.

Mårten Simonsson holds a MSc E.E. from the Royal Institute of Technology (KTH). He is currently a PhD Student at the Department of Industrial Information and Control Systems at KTH in Stockholm, Sweden. He focuses his research on the area of IT governance and the overall goal is to improve the COBIT framework to better support maturity assessment of IT organizations. Mårten is also active as a consultant in the area of enterprise architecture and IT governance.
Chapter 3

Paper B: Analyzing System Maintainability using Enterprise Architecture Models
ANALYZING SYSTEM MAINTAINABILITY USING ENTERPRISE ARCHITECTURE MODELS

By Robert Lagerström

Abstract
A fast and continuously changing business environment demands flexible software systems easy to modify and maintain. Due to the extent of interconnection between systems and the internal quality of each system many IT decision-makers find it difficult predicting the effort of making changes to their systems. To aid IT-decision makers in making better decisions regarding what modifications to make to their systems, this article proposes extended influence diagrams and enterprise architecture models for maintainability analysis. A framework for assessing maintainability using enterprise architecture models is presented and the approach is illustrated by a fictional example decision situation.

Keywords
enterprise architecture analysis, extended influence diagrams, system maintainability, system modeling

INTRODUCTION
A fast and continuously changing business environment demands flexible software systems easy to modify and maintain. Today many systems at an enterprise are interconnected and a change to one system may cause a ripple effect among the other systems. Numerous systems have also been developed and modified during several years and to make further changes to them demands a lot of effort from the organization, this poses questions such as. Is there enough documentation describing the systems and has the documentation been updated correctly after each change? Is the source code commented and written with good design in mind? These are some of the questions an IT-decision maker faces when wanting to implement changes to systems. It would be useful for IT-decision makers to be able to analyze how much effort a certain change to a system would require, but in a large enterprise it can be difficult to assess this effort since there are often many systems, persons, and processes involved in a change.

During the last decade, enterprise architecture (EA) has grown into an established approach for software system management. EA is model based in the sense that diagrammatic descriptions of the systems and their environment constitute the core of the approach. Some EA models contain information about the maintainability of a system, thus one could use EA models for system maintainability analysis.

This article proposes an approach for system maintainability analysis using EA models and extended influence diagrams. The maintainability framework presented in this article, in the form of an extended influence diagram, consists of attributes affecting the maintainability of software systems. The values of the attributes in the framework can be gathered from EA models and with the mathematics in extended influence diagrams a quantitative value of the maintainability effort can be calculated and used in decision situations regarding what modifications to make to a system. Figure 1 on the next page presents this approach graphically; the analysis part is depicted as a machine, containing an extended influence diagram, receiving EA models as input and presenting the analysis result as output.
This article unfolds as follows: Section 2 contains information about how to perform EA analysis using extended influence diagrams. Then the framework for system maintainability is presented in Section 3, the framework consists of an extended influence diagram and a metamodel. In Section 4 an example analysis is demonstrated using the proposed maintainability framework. Finally, the article is concluded in Section 5.

ENTERPRISE ARCHITECTURE ANALYSIS

The purpose of having EA models and conducting analysis of these is to facilitate rational decision making regarding software systems at an enterprise. In the article Extended Influence Diagrams for Enterprise Architecture Analysis by Johnson et al. (2006) a language called extended influence diagrams was introduced as a way of modeling goals, like maintainability, for EA analysis.

Extended influence diagrams graphically represent a network of nodes and relations. The different relations that can be used in an extended influence diagram are either causal, informational, or definitional relations. The nodes represent variables, and there are three different types of nodes; decision nodes, chance nodes, and utility nodes. Decision nodes represent the decisions that can be made, e.g. selecting between different systems scenarios. Utility nodes represent the goals, e.g. system maintainability. Chance nodes represent all variables related to the utility node, e.g. system size and system complexity. The syntax for the different relations and nodes is presented in Figure 1 on the next page. Extended influence diagrams are an extension of influence diagrams, thus inheriting the Bayesian mathematics. More information on Influence diagrams can be found in Howard (1983) or Shachter (1988). The nodes in extended influence diagrams have conditional probability matrices related to them, these matrices represent the states each node can have and the probability of each state occurring. An example extended influence diagram with a conditional probability matrix is presented in Figure 2 on the next page adjacent to the extended influence diagram syntax. In this example the probability matrix represents the probabilities of the attribute System Size to be small, medium, or large if a Scenario X or a Scenario Y is selected. As can be seen in Figure 2, the System Size is medium with a probability 0.8 (80%) if Scenario X is selected.
In the approach of using extended influence diagrams for EA analysis, an extended influence diagram is related to enterprise architecture models through the entities, sometimes referred to as classes, and the attributes in the models. This relation between an extended influence diagram and an EA model is visualized in Figure 3. What entities and attributes a model should contain to enable different analyses is determined by the extended influence diagram one chooses to use for a certain analysis. The utility node in the extended influence diagram represents the goal of the analysis, e.g. information security, performance, or maintainability. When the analysis goal is set, i.e. the extended influence diagram is selected, each node in the extended influence diagram relates to one attribute in the model (see Figure 3).

There are several meta-models, often called viewpoints, used in enterprise architecture modeling. For example, one could use the meta-models presented by Cummins (2002), O’Rourke (2003), Lankhorst (2005), and Niemann (2006). Common for all meta-models is that they contain entities, but what is often missing are the attributes of these entities. Lankhorst (2005) presents a viewpoint called infrastructure usage viewpoint which contains the entities hardware, supporting system software, networks, applications, and services. There are no attributes for these entities presented, e.g. the network entity could have the attribute performance or the supporting system software entity could have the attribute number of lines of source code. It is proposed by Johnson et al. (2006) that the EA meta-models should contain both entities and attributes, and that the attribute values should be presented in the form of conditional probability matrices like the nodes in the extended influence diagrams, thus forming a relation between the extended influence diagram and the model. The relation between models and extended influence diagrams is visualized in Figure 3 below. Here, an EA model of a System Scenario X has an entity called System A. This entity in turn has the attribute Size. The values of the attribute Size of System A is presented in the form of a conditional probability matrix, this matrix forms the connection to the node called System Size in the extended influence diagram.
MAINTAINABILITY FRAMEWORK

In this section the approach of analyzing different system qualities is applied with extended influence diagrams and EA models on the system quality maintainability, including the extended influence diagram for maintainability and the meta-model used when modeling for maintainability analysis.

Extended Influence Diagram For System Maintainability

System maintainability is defined as the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment (IEEE, 1990). To simplify assessments of maintainability researchers and practitioners have been looking for the attributes affecting maintainability for a long time and several frameworks have been proposed, such as Oman (1992), Chan (1996), Granja-Alvarez (1997), ISO (2001), Aggarwal (2002), and Matinlassi (2003). The maintainability framework presented in this article is to a large extent influenced by the work of Oman (1992).

System maintainability is said to be affected by; the maturity of the system’s development and maintenance personnel, the maturity of the development and maintenance processes, the quality of the system’s supporting documentation, the system’s architectural quality, the platform’s quality on which the system executes, and the system’s source code quality. Since these variables are difficult to measure, they have been further broken down into more easily measurable attributes. For example the maturity of the system’s development and maintenance personnel is measured in the staff’s level of experience with development and maintenance work, the staff’s level of language expertise on the programming languages used within the system, and finally the staff’s level of knowledge on the system they are maintaining. In Figure 4 below, the extended influence diagram for system maintainability is presented, further information on the framework can be found in Johnson (2007).
Meta-Model for System Maintainability Analysis

The meta-model used when modeling for system maintainability analysis should be able to represent the information in the maintainability extended influence diagram, i.e. the values of all nodes in the extended influence diagram affected by the decision node should be collected from the models that are based on the maintainability meta-model. The maintainability extended influence diagram presented in Figure 5 below has nodes related to persons, processes, documents, systems, platforms, and source code. Thus these are the entities needed in the meta-model. Looking at different meta-models proposed by other authors it is clear that the entities needed in a maintainability meta-model could be found in several already existing meta-models. On the contrary, the attributes for each entity matching the nodes in the extended influence diagram are more difficult to find in others work, e.g. that the entity Source Code has the attribute Level of Coupling. Because of this a new meta-model was developed, this meta-model is influenced by already existing meta-models but with the addition of attributes for the entities.

Figure 5 depicts the meta-model, which has six entities and each entity has several attributes all related to nodes in the extended influence diagram for maintainability.
MAINTAINABILITY EXAMPLE ANALYSIS

This section presents an example analysis employing the approach of EA analysis using the maintainability extended influence diagram and meta-model. Firstly, the fictional enterprise and its IT decision-maker are introduced, and then the decision situation and its future system change scenarios are modeled. Finally, the analysis and results of the example are presented.

Enterprise and Information Technology Decision-Maker Background

As a result of almost a decade of mergers and acquisitions on the deregulated European electricity market, Southern Energy is today one of Europe’s largest energy companies, acting in all parts of the electricity value chain. The company has a large amount of IT systems required to ensure a non interrupted flow of electricity within the Southern Energy’s distribution network. There is a dedicated CIO making decisions related to IT, the CIO is constantly involved in, and supervises, the evolution of the enterprise’s information systems. He has two enterprise architects working within the company under his command. The enterprise architects spend a considerable amount of time each week on continuous updating of the models that describe the enterprise architecture and also assisting the CIO in analysis of the models’ content.

Southern Energy’s geospatial network system, GeoNet needs some modifications due to changed requirements. The CIO has two changes he wants to implement, but his budget only allows him to do one. The first change scenario that the CIO would like to implement is to move the facilities record application in GeoNet to an independent database. The second change is moving the grid calculations application to a custom made calculation system. The CIO would like to analyze the effort of making these two changes, and compare the effort between the two scenarios so he could choose the scenario requiring the least effort. The CIO and his two enterprise architects chooses to use the maintainability extended influence diagram and meta-model proposed in this article. Based on the maintainability meta-model, the CIO selects the EA models that can be employed in order to assess the maintainability. The enterprise architects needed to add some information in the models and thus had to collect some data regarding the system GeoNet and its environment. They found some documentation regarding the system and also performed some interviews with maintenance and development personnel. The enterprise architecture model of the first change scenario for GeoNet is presented in the following subsection.

Change Scenario 1: Moving the facilities record application from GeoNet

In the first scenario, moving the facilities record application in GeoNet to an independent database, the enterprise architects collected information related to this change based on the
attributes in the meta-model. The enterprise architects found that the GeoNet system executes on a Linux platform and that the facilities record application source code is compiled into the application affected by the change. Further they found that there is a process for maintaining the facilities application and that Mr. Anderson is the person making changes to this application. There is also a design specification related to the facilities record application. Further the architects found that GeoNet exchanges data with several other systems; a network calculation system, an ERP system, a CRM system, and a SCADA system.

Data regarding the attributes in the meta-model was gathered by the enterprise architects and inserted in the model. The data was accordingly also inserted into the extended influence diagram to calculate the maintainability, i.e. the effort of implementing the change. For example the attribute Platform Quality was calculated from the attributes Availability, Porting Tools, and Level of Standardization. The Availability attribute was measured by the enterprise architects to be Low with the probability 0.1, Medium with the probability 0.8, and High with the probability 0.1. This means that the availability of the platform is Low with 10 % certainty, Medium with 80 % certainty, and High with 10 % certainty. Based on the three measured and modeled attributes of the platform, its quality was calculated in the extended influence diagram with the inherent Bayesian mathematics. The quality of GeoNet’s Linux platform was calculated to be Low with the probability 0, Medium with the probability 0.84, and High with the probability 0.16.

![Extended Influence Diagram](image)

Figure 6. The EA model of the affected parts of the GeoNet system in Scenario 1.

**Change scenario 2: Moving the grid calculations application from GeoNet**

In the second scenario, moving the grid calculations application to a custom made calculation system, the CIO’s enterprise architects collected similar data as in the first scenario and created a model using the meta-model for maintainability. The similarities between the two scenarios are the attributes of the system GeoNet’s architecture and the Linux platform it executes on. In the second scenario there is another maintenance process related to the calculation application, and there are two resources related to this process, Mrs. Williams and Mr. Smith. There is only one document relevant for the assessment of the second scenario, the calculation application design specification. Further there are two different source code files, one for the calculation application and one for the calculation presentation application. The collected information about the second scenario and the calculated values of the attributes were presented in an enterprise architecture model like the one for the first scenario.

**Maintainability analysis of the two change scenarios**

Using software for Bayesian analysis such as Genie (2007) or Hugin (2007), the
The maintainability of the two scenarios described in the above subsections was calculated and compared. The maintainability of Scenario 1 was evaluated to be Low with the probability 0, Medium with the probability 0.107, and High with the probability 0.893. The maintainability of the second scenario was calculated to be Low with the probability 0, Medium with the probability 0.793, and High with the probability 0.207. The results are presented in Figure 7. The first diagram in the figure visualizes the expected mean values of the scenarios; where Scenario 1 is calculated to 95 % and Scenario 2 to 60 %. Since the collected data is not 100 % certain, there is also an uncertainty related to the scenario values, this is visualized as thin bars in the first of the three diagrams. For more information on the calculations the interested reader is referred to (Howard, 1983; Johnson, 2007; and Shachter, 1988).

The results from the two change scenario evaluations can now be compared to aid the CIO of Southern Energy in selecting the change requiring the least effort. In Figure 7 three diagrams visualize the results of the maintainability analysis; the first diagram presents the comparison of the two scenarios including the uncertainty of the assessment, the second diagram presents the probabilities of Scenario 1’s values, and the third diagram presents the probabilities of Scenario 2’s values.

![Figure 7](image-url)

Figure 7. Results; the first diagram is a comparison between the two scenarios, the second diagram contains the probabilities of the maintainability in Scenario1 being low, medium and high, and the third diagram provides the probabilities of the maintainability in Scenario 2 being low, medium, and high.

The results presented provide support for the CIO in the choice between the two scenarios. As can be seen in the diagrams the CIO should make the choice of implementing change scenario one, to move the facilities record application in GeoNet to an independent database. In the first of the three diagrams there is a level of uncertainty associated with both scenarios’ maintainability. The uncertainty of the results could be decreased by engaging in more data collection, i.e. by increasing the precision of the enterprise architecture models. This would provide better decision support for the CIO but on the other hand require more resources.

CONCLUSIONS

This article proposes the approach of employing extended influence diagrams and enterprise architecture models when analyzing system maintainability. A framework for maintainability was provided, in the form of an extended influence diagram and a meta-model to be used when modeling systems for maintainability analysis. The extended influence diagram and the meta-model were then used in a fictional example to illustrate how these can aid in decision situations.
REFERENCES


Chapter 4

Paper C: A Method for Creating Enterprise Architecture Metamodels – Applied to Systems Modifiability Analysis
A METHOD FOR CREATING ENTERPRISE ARCHITECTURE METAMODELS - APPLIED TO SYSTEMS MODIFIABILITY ANALYSIS

ROBERT LAGERSTRÖM
ULRIK FRANKE
PONTUS JOHNSON
JOHAN ULLBERG

Industrial Information and Control Systems, Royal Institute of Technology
SE-100 44 Stockholm, Sweden
{roberl, ulrikf, pj101, johanu}@ics.kth.se

Enterprise architecture models can be used in order to increase the general understanding of enterprise systems and specifically to perform various kinds of analysis. It is generally understood that such modeling encompasses general scientific issues, but the monetary aspects of the modeling of software systems and their environment are not equally well acknowledged. Even more so, creating a good metamodel for enterprise software systems analysis is an important but challenging task. The present paper describes a method for creating metamodels for such analysis. The enterprise architecture models are formalized using probabilistic relational models, which enables the combination of regular entity-relationship modeling aspects with means to perform enterprise architecture analysis. The proposed method for creating metamodels is general, however this paper presents the method by creating a metamodel for systems modifiability, i.e. the cost of making changes to enterprise-wide systems. The method and the method outcome, i.e. the metamodel, is validated based on survey and workshop data and the applicability of the metamodel is illustrated with an instantiated architectural model based on a software change project at a large Nordic software and hardware vendor.

Keywords: Enterprise architecture; Metamodelling; Probabilistic relational models; Software modifiability.

1. Introduction

Managing software systems today is a complex business. In order to achieve effective and efficient management of the software system landscape it is essential to be able to assess the current status of system properties such as availability, performance, security, and modifiability, as well as predict their values in different future scenarios. Estimation of these properties is however a great challenge that to a large extent can be addressed by introducing relevant models as a means of abstraction, which can be achieved with enterprise architecture modeling. The purpose of this paper is to propose a method for creating enterprise architecture metamodels that
support analysis of various system properties.

1.1. Enterprise architecture

In recent years, Enterprise Architecture (EA) has become an established discipline for business and software system management [Ross et al. 2006]. Architecture models constitute the core of the approach and serve the purpose of making the complexities of the real world understandable and manageable to humans [Winter and Fischer 2007]. Enterprise architecture ideally aid the stakeholders of the enterprise to effectively plan, design, document, and communicate IT and business related issues, i.e. they provide decision support for the stakeholders [Kurpjuweit and Winter 2007].

A key underlying assumption of the EA models is that they should provide some more aggregated knowledge than what was merely put into the model in the first place. Software system architecture, for instance, does not only keep track of the set of systems in an enterprise and their internal relationships, it also provide information about the dependencies between the systems. Conclusions can be drawn about the consequences in the enterprise given that one specific system is unavailable. The enabling of this type of analysis is extremely important in providing value of EA for its stakeholders. Unfortunately however, EA frameworks rarely explicitly state neither what kinds of analyses that can be performed given a certain model nor the details on how the analysis should be performed [Johnson et al. 2007; Franke et al. 2009b].

Another permeating problem in EA modeling is the uncertainty that is related to the model [Johnson et al. 2007]. For instance, whether a model is the result of a very thorough and recent investigation or a quick read-through of somewhat old documents will impact the quality of the decision support that the model offers to its stakeholders. Are all the software systems in the model still in use, is the data flow still as depicted, and does the process structure really illustrate what is actually happening? This kind of uncertainty is not addressed by EA frameworks of today. Again the user of the models is simply left to her best knowledge or gut feeling when estimating to what extent the EA model, and the analyses based on it, can and should be trusted.

This paper presents a formalized approach to enable analysis of EA models by connecting the analysis to the language used to express the models, the metamodel. In other words, the approach devises analysis frameworks in terms of metamodels suitable for analysis of different properties such as modifiability and security. Specifically, the focus of the approach is to identify and manage the problems that arise when such metamodels are created. The approach also copes with empirical uncertainties in the application of the devised analysis framework by not considering the information in EA models as fixed constants but rather as probabilities. The underlying fundamental formalism in the approach is called Probabilistic Relational Models (PRMs) [Friedman et al. 1999].
1.2. Enterprise system modifiability

As discussed in the previous subsection, enterprise architecture models can be used to analyze different system properties and provide information for the decision maker regarding different scenarios. In this paper enterprise software system modifiability, i.e. the cost of making changes to enterprise-wide software systems, will be considered as a running case to illustrate the proposed metamodel creation method.

Business environments today progress and change rapidly to keep up with evolving markets. Most business processes are supported by software systems and as the business processes change, the systems need to be modified in order to continue supporting the processes. Modifications include extending, deleting, adapting, and restructuring the enterprise systems [Bass et al. 1998]. The modification effort range from adding a functional requirement in a single system to implementing a service oriented architecture for the whole enterprise.

An essential issue with today’s software systems is that many of them are interconnected, thus a modification to one system may cause a ripple effect among other systems. Also, numerous systems have been developed and modified during many years. Making further changes to these systems might require a lot of effort from the organization, for example due to a large amount of previous modifications implemented ad hoc. Problems like these pose questions for IT decision makers such as: Is there enough documentation describing the systems, and has the documentation been updated correctly after each modification? Is the source code easy to understand? Which systems are interconnected?

Several studies show that the modification work is the phase that consumes the greatest portion of resources; Harrison and Cook reports that over 70 % of the software budget is spent on maintenance [Harrison and Cook 1990], Pigoski refers to studies stating that the maintenance cost, relative to the total life cycle cost of a software system, has been increasing from 40 % in the early 1970s up to 90 % in the early 1990s [Pigoski 1997], and Jarzabek states that “the cost of maintenance, rather than dropping, is on the increase” [Jarzabek 2007].

The activities of modifying enterprise systems are typically executed in projects, and IT decision makers often find it difficult to predict and plan their change projects. Thus, a large proportion of the projects aiming to modify a software system environment fail. That is, the projects tend to take longer time and cost more than expected. Laird and Brennan states that 23 % of the software projects are cancelled before completion, whereas of those completed only 28 % were delivered on time, and the average software project overrun the budget by 45 % [Laird and Brennan 2006]. This can often occur due to lack of information about the systems being changed. According to Laird and Brennan, software engineers must be able to understand and predict the activities, as well as manage the risks, through estimation and measurement [Laird and Brennan 2006]. Therefore, it would be useful for IT decision makers to gather more information in a structured manner and use this information to analyze how much effort a certain modification to an enterprise
software system would require. This paper will address these issues of software change by employing enterprise systems modifiability analysis as a running case, thus providing a metamodel for assessment of software change project cost and manage the problems that occur while devising such a metamodel.

1.3. Creating enterprise architecture metamodels

Metamodels are the core of enterprise architecture. They describe the fundamental artifacts of business and IT. Such high level models provide a common language and a clear view on the structure of and dependencies between relevant parts of the organization [Winter and Fischer 2007]. However, when creating a metamodel for enterprise architecture analysis there are some aspects that need to be considered. First, a great number of factors influence system properties. Second, the factors are intertwined in a complex manner. The researcher or practitioner who sets out to model these interdependencies thus inevitably faces a discomforting amount of modeling choices, all of which to some extent influence the ability of the final assessment framework to provide accurate decision support for management decisions. Furthermore, all modeling choices represent a cost in terms of collecting the information needed for actually using the model. This cost, whether expressed in money, effort, or time, must be kept under control, lest the entire modeling effort be misguided.

This paper delineates a method for creating enterprise architecture metamodels for analysis. While the focus is assessment of software modification projects, the method itself aims to be general. The method and the resulting metamodels address the concerns mentioned above. As all EA frameworks, ours aim to clarify complex interdependencies, but is at the same time subject to uncertainties. As with all estimates, ours are built upon basic scientific questions of measurement, scales and precision. As with all metamodels, ours requires careful consideration of the classes and relationships included.

A number of general problems, outlined in section 2, serve as a roadmap for the rest of this paper. The theoretical considerations necessary for creating a metamodel presented can all be related to this roadmap. In that sense, it provides a unifying framework, explaining how the different parts together constitute a single whole. In a previous paper, a formal argument was given [Franke et al. 2009c] for the possibility to treat all the roadmap problems jointly. In this paper, we expand the treatment slightly, putting it into the broader context of an entire EA approach, rather than single software measures considered by themselves.

1.4. Outline

The remainder of the paper is structured as follows: Section 2 introduces general modeling problems for enterprise architecture analysis. Section 3 presents the probabilistic relational models which serve as the underlying formalism for the enterprise architecture metamodels proposed for analysis. The following section proposes the
enterprise architecture metamodel creation method which includes a development process and guidelines supporting the creation of metamodels for analysis. Next, in section 5 an instantiated model for a software change project is described in order to show the applicability of the modifiability metamodel. Section 6 contains data for validation of the method by considering the correctness of the qualitative structure of the metamodel and by discussing the usability of the creation method. In section 7 related work is presented and finally, section 8 summarizes the paper with conclusions.

2. General modeling problems for enterprise architecture analysis

Fig. 1. A schematic overview of the modeling process when change cost analysis is the main purpose, displaying the five problems listed in the text.

Many system properties – availability, performance, security, and modifiability, to name a few – share the elusive feature that while they are easy to define \textit{a posteriori}, i.e. after system implementation, such definitions give precious little guidance on how to ensure them \textit{a priori}, i.e. before system implementation. For example, measuring the cost of change of a system \textit{a posteriori} is mere book-keeping. But assessing it beforehand is a formidable task. Such assessment must be carried out by measuring variables available prior to the modification. The literature, provides a wealth of different methods to make such cost of change assessments. The modeler however cannot afford to employ them all, she requires accurate and cost-efficient decision support. The same holds true for the assessment of other system properties, even though we shall use cost of change (modifiability) as a running case for the remainder of the present paper.

Fig. 1 is a generic depiction of a modeling process, aiming to create a framework for prediction of costs associated with software modification projects. Five key
problems, numbered in the figure, need to be addressed by such a framework:

(1) **Correlation and causality.**

(a) **Correlation.** The choice of an *a priori* measurement quantity is the problem of finding a measure that correlates accurately with the sought *a posteriori* quantity. A typical example of this question is given by the cost of change, i.e. the cost for modifying or amending a software program, which often is estimated by software complexity. Software complexity in itself is just a measure of certain properties of the program code. No complexity measures make any mention of the cost or time spent creating or modifying this code. Nevertheless, it is generally assumed that complex computer programs are difficult to maintain and modify. This is the correlation (and causal relation) that, supposedly, binds together the *a priori* metric of complexity with the *a posteriori* system property of cost of change.

(b) **Causality.** Causality is closely linked to correlation, but they are not the same, and there is an ever-present need to separate the one from the other. In the present context, we can discern two different decision-support aspects: First there is a *prediction* aspect, reflecting the need for *a priori* accurate measures of how things turn out *a posteriori*. To achieve this, correlation is sufficient. Second, however, there is an *action guiding* aspect, reflecting the need for control. Here, we take another step: seeing an *a priori* estimate of high project costs, how can we act so as to lower these costs? To achieve this, mere correlation is insufficient: we need causal relations, so that our actions causally impact and change the cost.

In the rest of this article, we try to separate correlation from causality, usually aiming to create models that are indeed causal, not merely exhibit correlations.

(2) **Uncertainties.** Uncertainty is inherent in any EA analysis. However, there are many kinds of uncertainties, and to better understand and alleviate them, they can be broken down into sub-categories. The following list is based on [Johnson et al. 2007]:

(a) **Definitional uncertainty.** Most concepts can be interpreted in many different ways. One example is the notion of software complexity, that might be interpreted as for instance Halstead complexity, cyclomatic complexity, or simply lines of code. Since there is no general agreement on the meaning of all concepts, it is imperative that assessment frameworks are explicit about the intended meaning of the terms used.

(b) **Theoretical heterogeneity.** There is no general agreement on a single framework within which all EA knowledge can be expressed. Despite attempts to rectify this, such as [Franke et al. 2009b; International Standardization Organization/International Electrotechnical Committee 2007], this is nevertheless likely to remain the case for the foreseeable future.
(c) **Causal uncertainty.** It is rare that empirical phenomena are fully understood. For example, it is unlikely that an IT decision maker knows for certain to what extent the introduction of a new tool, a new training procedure, or the adoption of a more mature change management process will increase modifiability.

(d) **Empirical uncertainty.** Whenever architecture models of an enterprise are created, there is a risk of erroneous modeling. Incorrect modeling information can easily arise due to lack of time or manpower, immature processes, or simple mistakes. Furthermore, the information might have been correct when collected, but has become obsolete by the time of its use.

As is concluded in [Johnson et al. 2007], these uncertainties call for a probabilistic analysis framework. Section 3 describes the formalism used throughout this paper.

(3) **Measurement devices.** Representational theory of measurement is about mapping an empirical relational system, i.e. observable reality, into a mathematical model, a numerical relational system [Hand 1996]. As put in [Nagel 1931], measurement is "the correlation with numbers of entities which are not numbers". This correlation aspect of measurement is one of our foremost interests for the purpose of this paper. To the modeler, the use of a particular measuring device will affect the model’s theoretical performance by its inherent accuracy limits.

One of the most convenient methods for collecting information about certain characteristics of a system is to consult an expert in the field. In this paper, we consider the use of expert estimation to be a kind of measurement, on a par with other methods, such as using measurement software.

(4) **The selection of appropriate scales.** That the choice of scales has non-trivial repercussions on scientific models has been known at least since Stevens’ seminal 1946 paper [Stevens 1946]. The fact that different scales permit different statistical operations creates constraints on the permissible structure of assessment frameworks. Following our example, we note that interval scale concepts such as LoC, Halstead and cyclomatic complexity might be transformed onto ordinal scales when discretized. One such possible ordinal scale is High, Medium, Low. The loss of information is evident, but often unavoidable for instance for expert estimates.

(5) **Discretization of measurement variables.** A special case of the preceding problem is discretization of properties that are continuous. In the context of measurement, discretization not only approximates a continuous phenomenon, but also enables mapping onto a predetermined scale. In general, of course, such mappings entail a loss of information. The general trade-off thus amounts to achieving the simplifications of discretization while retaining as much information as is necessary in a given context.

Throughout this paper, we will address the problems above, using them as start-
ing points in the process of creating the enterprise architecture metamodels for analysis. As it turns out, the correlation, causality, and uncertainty problems lend themselves to a more general analysis, whereas the measurement devices, scales and discretization need to be addressed more on a case by case basis. In the following, thorough examples from both categories are given.

3. Probabilistic relational models

As stated in the introduction Probabilistic Relational Models (PRMs) serve as the underlying formalism for the enterprise architecture metamodels proposed for analysis.

A PRM [Friedman et al. 1999] specifies a template for a probability distribution over an architecture model. The template describes the metamodel for the architecture model, and the probabilistic dependencies between attributes of the architecture objects. A PRM, together with an instantiated architecture model of specific objects and relations, defines a probability distribution over the attributes of the objects. The probability distribution can be used to infer the values of unknown attributes, given evidence of the values of a set of known attributes.

An architecture metamodel $\mathcal{M}$ describes a set of classes $\mathcal{X} = X_1, \ldots, X_n$. Each class is associated with a set of descriptive attributes and a set of reference slots. The set of descriptive attributes of a class $X$ is denoted $A(X)$. Attribute $A$ of class $X$ is denoted $X.A$ and its domain of values is denoted $V(X.A)$. For example, a class System might have the descriptive attribute Size, with domain \{large, medium, small\}. The set of reference slots of a class $X$ is denoted $R(X)$. We use $X.\rho$ to denote the reference slot $\rho$ of $X$. Each reference slot $\rho$ is typed with the domain type $\text{Dom}[\rho] = X$ and the range type $\text{Range}[\rho] = Y$, where $Y \in \mathcal{X}$. A slot $\rho$ denotes a relation from $X$ to $Y$ in a similar way as Entity-Relationship diagrams. For example, we might have a class Documentation with the reference slot Describes whose range is the class System.

An architecture instantiation $\mathcal{I}$ (or an architecture model) specifies the set of objects in each class $X$, the values for the attributes, and the reference slots of the objects. For example, Fig. 8 presents an instantiation of the change project metamodel of Fig. 6. It specifies a particular set of changes, systems, documents, etc., along with values for each of their attributes and references. For future use, we also define a relational skeleton $\sigma_r$ as a partial instantiation which specifies the set of objects in all classes as well as all the reference slot values, but not the attribute values.

A probabilistic relational model $\Pi$ specifies a probability distribution over all instantiations $\mathcal{I}$ of the metamodel $\mathcal{M}$. This probability distribution is specified as a Bayesian network [Jensen 2001], which consists of a qualitative dependency structure and associated quantitative parameters.

The qualitative dependency structure is defined by associating with each attribute $X.A$ a set of parents $Pa(X.A)$. Each parent of $X.A$ has the form $X.\tau.B$ where
A method for creating enterprise architecture metamodels – applied to systems modifiability analysis

$B \in A(X, \tau)$ and $\tau$ is either empty, a single slot $\rho$ or a sequence of slots $\rho_1, \ldots, \rho_k$ such that for all $i$, $\text{Range}[\rho_i] = \text{Dom}[\rho_{i+1}]$. For example, the attribute $\text{Cost}$ of class $\text{ChangeProject}$ may have as parent $\text{ChangeOrganization.Developer.Expertise}$, thus indicating that the cost of a prospective software modification project depends on the expertise of the developers employed in the organisation. Note that $X, \tau, B$ may reference a set of attributes rather than a single one. In these cases, we let $x.A$ depend probabilistically on some aggregate property over those attributes, such as the logical operations $\text{AND}$, $\text{OR}$, and $\text{NOR}$. In this paper we use the arithmetic operations $\text{SUM}$ and $\text{MEAN}$ as aggregate functions. For instance, if there are several developers engaged in a modification project, we might aggregate the individual developers’ expertise into a mean expertise of the whole development team.

Considering the quantitative part of the PRM, given a set of parents for an attribute, we can define a local probability model by associating a conditional probability distribution (CPD) with the attribute, $P(X.A|Pa(X.A))$. For instance, $P(\text{ChangeProject.Cost} = \text{high}|\text{ChangeOrganization.Developer.Expertise} = \text{low}) = 90\%$ specifies the probability that the project cost will be high, given the expertise of the developers involved.

We can now define a PRM $\Pi$ for a metamodel $M$ as follows. For each class $X \in X$ and each descriptive attribute $A \in A(X)$, we have a set of parents $Pa(X.A)$, and a conditional probability distribution (CPD) that represents $P_{\Pi}(X.A|Pa(X.A))$.

Given a relational skeleton $\sigma_r$ (i.e. a metamodel instantiated to all but the attribute values), a PRM $\Pi$ specifies a probability distribution over a set of instantiations $I$ consistent with $\sigma_r$:

$$P(I|\sigma_r, \Pi) = \prod_{x \in \sigma_r(X)} \prod_{A \in A(x)} P(x.A|Pa(x.A))$$

where $\sigma_r(X)$ are the objects of each class as specified by the relational skeleton $\sigma_r$.

A PRM thus constitutes a formal machinery for calculating the probabilities of various architecture instantiations. This allows us to infer the probability that a certain attribute assumes a specific value, given some (possibly incomplete) evidence of the rest of the architecture instantiation. In addition to expressing and inferring uncertainty about attribute values as specified above, PRMs also provide support for specifying uncertainty about the structure of the instantiations. A more detailed description of this topic is, however, beyond the scope of the paper.

4. The enterprise architecture metamodel creation method

Creating a good metamodel (probabilistic relational model) is not trivial. Obviously, it is important that the metamodel is tailored for the management tasks it should support, i.e. what kind of analysis the metamodel is intended to support. For instance, if one seeks to employ an enterprise architecture model for evaluating business process efficiency, the information required from the model differs radically from the case when the model is used to evaluate the modifiability of an enterprise software system.
Addressing this task, the general problems presented in section 2 are used as an underlying framework. In this section, our main focus will be on the first problems; correlation, causality and uncertainties. The issues of measurement devices, scales, and discretization will be addressed subsequently, both in section 4.1.4 and in section 5. All the problems, of course, need to be considered when creating and using a metamodel for enterprise architecture analysis.

4.1. Methods for creating the qualitative part of a metamodel

This subsection presents the method steps for creating the qualitative part of a metamodel, i.e. the classes, reference slots, attributes and their parents.

The basic modeling aim is to ensure the correlation between \emph{a priori} measurements and \emph{a posteriori} properties of interest. For any such property, there are numerous candidates for \emph{a priori} measurements, each with its own set of advantages and disadvantages.

Our running case makes this even clearer: the cost of change, i.e. the cost for modifying or amending a software system, can be and is often estimated by the software complexity, as discussed in section 2. However, taking complexity alone as a sufficient indicator of the cost of change is usually not enough. Most estimation methods available indeed use multiple \emph{a priori} measures.

The first step of the method for creating decision support metamodels is thus to find a set of appropriate \emph{a priori} measures for change cost estimation and decision making, i.e. finding measures with high correlation and causal influence on change cost. This can be done in several ways, for instance by studying research literature, doing experiments and case studies, or using expert opinions. In subsequent iterations, variables causally affecting the variables found in the former iterations are identified. This iterative process continues until all paths of variables, and causal relations between them, have been broken down into variables that are directly controllable by the decision maker, see part 1 of Fig. 2. For instance, it might be identified through literature study that the goal \emph{decrease change project cost} is affected by \emph{enterprise system understandability} and \emph{change management process maturity}. In the next iteration the \emph{enterprise system understandability} might be affected by the, to the IT decision maker, directly controllable variables \emph{system size} and \emph{system internal coupling}. This iterative goal break down structure approach has also been presented in [Lagerström et al. 2009e] as a part of a method to create stakeholder-oriented metamodels.

4.1.1. Eliciting variables for goal break-down fragments

The goal to be broken down in our running case is enterprise system modifiability, i.e. the cost of making changes to enterprise wide systems. The metamodel for modifiability analysis needs to have a high degree of construct validity in the sense that it actually provides good estimates and decision support for the goal variable under analysis. A convenient way of ascertaining a sufficient degree of construct
validity is to base the information in the goal break-down structure on scientific theory. Such theory is readily available in the plethora of scientific papers and articles published in scientific journals or presented at academic conferences. As we shall see, the use of scientific publications ensures that the models built reflect causal relations, i.e. handles the first problem in section 2. However, re-use of such previously published information also poses some challenges when it comes to the uncertainty problem. This is addressed in section 4.1.2 and 4.1.3.

[Lagerström et al. 2007] presents a method focusing on knowledge elicitation from scientific texts, i.e. finding the appropriate measures for a certain analysis, see part 2 of Fig. 2. In the example case it means breaking down the goal enterprise systems modifiability by studying research on this particular topic.

(1) Step one: select scientific text to use as a source for the goal break-down and identify which variable to break-down. Once the variable has been identified,
this is used as the variable under consideration in the first iteration of the process.

(2) Step two: Read the scientific text and identify all occurrences of the variable under consideration where a causal or definitional relation to other variables is implied. Extract these occurrences into an evidence database, a piece of evidence could be a sentence, a figure, a table, or combinations thereof.

(3) Step three: For each piece of evidence in the evidence database, identify if it is a causal relation or a definitional relation and between which variables this relation acts. Document this with an appropriate goal break-down syntax, e.g. extended influence diagrams as proposed in [Johnson et al. 2007].

(4) Step four: Use the variable fragments to identify variables related to the variable under consideration. Use each one of the identified variables as the new variable under consideration and iterate starting with step 2 in the process. Perform this iteration until no more variables related to the variable under consideration are found in the selected source.

(5) Step five: Perform sanity check of the variable fragments. Before this step, the created fragments give a fairly good representation of what the text actually states. However, there is always a chance that this representation does not accurately reflect the semantics of the chosen scientific text. In order to reflect the semantics, the user of the process is allowed to make some interpretations, provided that these interpretations are carefully documented. During this step, relations and variables that are implied in the source may be introduced. Moreover, relations and variables that are explicitly stated in the text, but that make no sense may be removed. During this step the variables in the goal break-down fragments need to be tested regarding controllability and if these are well defined or not. This process is further elaborated in subsection 4.1.2.

The guidelines presented above can be slightly altered and for instance be used when interviewing experts instead of or as an alternative to written scientific sources.

4.1.2. Expanding goal break-down fragments

In [Johnson et al. 2007] a process for managing extended influence diagrams, i.e. formalized causal goal break-down structures, is proposed cf. part 3 of Fig. 2. The presented process contains steps which together with the text elicitation guidelines provides a structured foundation when constructing a goal break-down fragment. The process presented aims to find variables for the goal break-down fragment which are well defined and controllable for the decision maker. This use of controllability constitutes an important way of ensuring appropriate causality for the action guiding aspect of the model, and of removing unnecessary correlations that do not possess this quality.

(1) The first step is to determine whether a variable in the goal break-down struc-
ture is lexically defined, stipulatively defined or undefined. A variable is lexically defined if we can assume that there is common agreement on its definition. Although almost all definitions can be challenged, concepts that would normally qualify as lexically defined include *weight in kilograms, number of processors* or *number of users*. For many concepts, however, such as *information security, architecture quality* and *competence* there is no universal definition even by very pragmatic standards. If a variable is not lexically defined, it needs to be stipulatively defined. As an example, the variable *documentation quality* might be defined by the two variables *documentation availability* and *documentation accuracy*. The stipulative definition of a variable results in the introduction of new variables into the goal break-down fragment. When new variables are introduced, this affects the conditional probability distributions of the child variables. These must therefore also be specified, thereby detailing the dependencies between the parents and the child.

(2) For each defined variable, the second step considers whether the variable is uncontrollable. An uncontrollable variable cannot at all be affected by the decision maker; this means that potential variations in the value of the uncontrollable variable are unrelated to the choices of the decision maker. If the variable is uncontrollable, it will not provide decision supporting information, so there is no need to continue exploring this branch of the goal break-down fragment. The variable can therefore be deleted.

(3) Step three queries whether the variable under consideration is semi-controllable. Semi-controllable variables are affected both by the decision maker’s choices and other, uncontrollable, phenomena. If a variable is semi-controllable, it is important to separate those aspects which are controllable from those which are not. Therefore, new variables are introduced into the goal break-down structure with causal relations to the semi-controllable variable under consideration. As an example, we might believe that the semi-controllable variable *mean time to repair* is causally affected by both the *maintainability* of the system and *flexibility* of working hour regulations. Of these, the maintainability might be controllable, while the working hour regulations might be beyond the decision maker’s domain of control.

(4) In the final step, the process considers whether the variables are directly or indirectly controllable. A directly controllable phenomenon is an immediate consequence of the decision maker’s choice. For instance, if a scenario is chosen where one system is replaced by another, this may directly entail that the *CPU speed* is increased. For variables that do not seem to be directly controllable, one or several new variables are introduced into the goal break-down structure, the focus shifts to the one of these, and the process returns to the beginning.

When the whole process is finished, the result is a goal break-down structure where the variables can be controlled by the decision maker. No variables are left undefined, and uncontrollable variables have no causal parents.
By using the goal break-down approach, including the knowledge elicitation guidelines and the control process, a causal goal break-down fragment is produced. Such a fragment is based on scientific knowledge, and exhibits variables that are all causally linked to a well-defined goal, controllable by the decision maker.

Fig. 3 presents a goal break-down fragment derived when decomposing the modifiability goal.

![Diagram](image)

Fig. 3. The outcome of the goal break-down method is a goal break-down fragment. This example fragment for modifiability is focusing on component understandability and variables causally related to this understandability.

### 4.1.3. Extracting metamodel elements

The goal break-down fragment with its well defined and controllable variables can now be translated to a metamodel fragment, cf. Fig. 4.

The set of variables and their relations have been identified in the goal break-down fragment, e.g. as in Fig. 3. Thus it is time to translate these into metamodel classes, attributes, and reference slots. A metamodel class can either represent physical artifacts, such as `computer` and `person`, or more conceptual ones such as `data flow` or `process`, depending on what the fragments of the goal break-down states. The attributes of the metamodel classes correspond to the variables found in the goal break-down part of the method, as presented in previous subsections.

Recall the goal break-down fragment presented in Fig. 3. This corresponds to the metamodel fragment presented in Fig. 5.

The method steps described so far will result in a number of goal break-down fragments and accompanying metamodel fragments, all based on the selected source or set of sources. However, depending on the granularity of the sources, these fragments are usually very small and local models, i.e. models describing the relations between a few select elements in great detail, but without a sense of a bigger picture. Furthermore, the fragments are sometimes completely disjoint, sometimes completely overlapping, and usually somewhere in between – all depending on the scope of the original sources. To make full use of the knowledge elicited, however, we need to merge the fragments into one metamodel. The challenge is to make sure that this metamodel remains coherent and non-ambiguous, despite its diverse
A method for creating enterprise architecture metamodels – applied to systems modifiability analysis

Fig. 4. Translating the goal break-down to a metamodel.

Fig. 5. The resulting metamodel fragment based on the modifiability goal break-down fragment example.

origins. This integration challenge is addressed in [Lagerström et al. 2008], which gives some guidelines for merging.

A metamodel fragment merge can be performed on several levels. First and foremost, classes can be merged. However, once classes have been merged, it is still an open question whether the attributes of the original classes are suitable for merging, or not. Therefore, the fragments at hand need to be examined in a number of dimensions. Some suitable criteria are those listed in Table 1.

<table>
<thead>
<tr>
<th>Class merge</th>
<th>Attribute merge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Similarity of names</td>
<td>Similarity of names</td>
</tr>
<tr>
<td>Similarity of sources</td>
<td>Similarity of sources</td>
</tr>
<tr>
<td>Similarity of reference slots</td>
<td>Similarity of attribute relations</td>
</tr>
<tr>
<td>Similarity of attributes</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Criteria for merging of metamodel fragment classes and attributes.
A few words of explanation are called for. The matter of name similarity is rather straightforward. To make assessment easier, the criterion can be categorized into identical, similar, or dissimilar names. Concerning sources, the question is whether the concepts come from the same place. If the authors, the interviewees, the companies, the scientific journals etc. are the same, this indicates that the classes or attributes are suitable for merging. The similarity of reference slots can be assessed according to whether the classes or attributes under consideration have all, some, or none of their reference slots to others in common. Finally, the similarity of the attributes of a class can be used as an additional heuristic for assessing whether the classes are suitable for merging.

This method can be implemented formally for machine execution, as described in [Lagerström et al. 2008], or used more informally for manual fragment integration. By using the guidelines for metamodel fragment merge, the metamodel depicted in Fig. 6 was finally obtained for modifiability analysis.

**Fig. 6.** The resulting modifiability metamodel after fragment merge.

### 4.1.4. Documenting metamodel definitions and scales

So far, our attention has been directed to the identification and break-down of relevant measures to support our decision-support endeavor. However, settling for a set of theoretical *a priori* measures is only a first metamodeling step. Data collection often requires concerted efforts from many investigators, who need a clear common picture of the concepts at hand. Similarly, the literature is often less than clear
on the exact nature of the concepts described or used in research. Both of these phenomena contribute to the importance of definitional uncertainty, as described in section 2. Definitional uncertainty must be at a minimum before precise analysis can be carried out.

At a first glance, the problem of clear definitions seems shallow. Having based the classes and attributes upon the scientific literature, it might be assumed that the precise meaning of the concepts used would be clear. This is, however, not necessarily the case. For instance, the complexity measure is commonly defined in several different ways, such as:

1. The Halstead measure which is based on the numbers of operators and operands, distinct as well as total, found in the code [Grubb and Takang 2003].
2. The number of lines of code (LoC), i.e. the software size, which is a crude estimate of complexity but nevertheless a possible choice. LoC is usually defined excluding comments and blank lines [Grubb and Takang 2003].
3. Cyclomatic complexity which was introduced by McCabe and is based on graph theory. It measures the number of linearly independent paths through a source code [Grubb and Takang 2003].

Even if every individual source is clear about its usage of the terms, it is important to keep in mind that the metamodel constructed in the previous steps represent the merging of several theoretical fragments. Such a merging is necessary to make the best use of the available knowledge, but for this reason alone it is clear that the final setting of definitions and scales is a task that necessarily falls on the shoulders of the metamodeler.

The metamodel creation method proposed here takes definitional uncertainty into consideration in several steps, as described in subsection 4.1.1 and 4.1.2. This subsection provides an example, based on the modifiability case, on how to document and describe the definitions found and used in the metamodel.

The metamodel presented in Fig. 6 contains a number of classes with accompanying attributes, these are all described in [Lagerström et al. 2009a,c]. In this paper the organizational view of the metamodel is more thoroughly presented, cf. Fig. 7. The organizational view focus on presenting the classes; change organization, architect team, and developer team. This means that the aggregating classes of architects and developers with their accompanying attributes also are described, as well as their relation to the architectural and component change activities. The following paragraphs present the organizational view of the modifiability metamodel in text and Table 2 contains the selected metamodel definitions and scales.

The change organization refers to the organization implementing the software system modifications, i.e., the parties involved in the project, such as the customer, the vendors, consultants etc. Attributes in the metamodel related to the change organization are number of architects and number of developers involved in the project, affecting one of the two types of activity costs each. [Grubb and Takang
Architects are the people in the change project who design and modify the architecture of the enterprise systems. Developers, on the other hand, are the ones working on the source code of the different components. The architects and developers both have the attributes expertise and time on project related to them. Expertise is stipulatively defined in terms of change project experience, source code/design language experience, and system experience. Time on project refers to the amount of time a person spend in the project compared to other parallel work. The team expertise is affected by the amount of time the team members spend on the project and the team expertise affect the activities cost. [Grubb and Takang 2003; Pigoski 1997; Boehm 1981]

Change projects can conceptually be divided into architectural change activities and component change activities. Architectural change activities are the activities concerning modifications on an architecture level, i.e. involving several systems or components, while component change activities concern modifications to a single component. Both types of change activities have the attributes cost and synchronization need, the former measured as number of man-hours and the latter as the amount of time spent on synchronization. The more systems, components, people involved, and the higher the coupling between them, the higher the need of synchronization among the different activities will be. [Boehm 1981; Grubb and Takang 2003]

4.2. Methods for creating the quantitative part of a metamodel

The former subsection presented the method steps for creating the qualitative part of an EA metamodel for analysis, i.e. the classes, attributes, and reference slots, cf. Fig. 6 for the resulting modifiability metamodel. This subsection presents the next step of the method, creating the quantitative part of such a metamodel. That is, to define the conditional probability distributions (CPDs) related to each attribute, as
A method for creating enterprise architecture metamodels – applied to systems modifiability analysis

<table>
<thead>
<tr>
<th>Class</th>
<th>Attribute</th>
<th>Definition</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change organization</td>
<td>No of architects</td>
<td>Number of persons in the change project with the assigned role architect</td>
<td>0,…,∞</td>
</tr>
<tr>
<td></td>
<td>No of developers</td>
<td>Number of persons in the change project with the assigned role developer</td>
<td>0,…,∞</td>
</tr>
<tr>
<td>Architect / Developer team</td>
<td>Expertise</td>
<td>The average expertise level of the team based on all members</td>
<td>High, Medium, Low</td>
</tr>
<tr>
<td></td>
<td>Time on project</td>
<td>Average time (in percentage) spent on the change project compared to other parallel work in total for all team members</td>
<td>0,…,100</td>
</tr>
<tr>
<td>Architect / Developer</td>
<td>Expertise</td>
<td>Stipulatively defined by change project, language, and system experience</td>
<td>High, Medium, Low</td>
</tr>
<tr>
<td></td>
<td>Change project experience</td>
<td>Number of years working with change</td>
<td>0,…,∞</td>
</tr>
<tr>
<td></td>
<td>Language experience</td>
<td>Number of years working with the, in this project, chosen design / programming language</td>
<td>0,…,∞</td>
</tr>
<tr>
<td></td>
<td>System experience</td>
<td>Number of years working with, the in this project, types of systems being changed</td>
<td>0,…,∞</td>
</tr>
<tr>
<td></td>
<td>Time on project</td>
<td>Time (in percentage) spent on the change project compared to other parallel work</td>
<td>0,…,100</td>
</tr>
<tr>
<td>Architectural / Component change activities</td>
<td>Cost</td>
<td>Total number of of man-hours spent on all change activities</td>
<td>High, Medium, Low</td>
</tr>
<tr>
<td></td>
<td>Synch. Need</td>
<td>Average percentage of time spent on synchronization between the change activities in the project</td>
<td>0,…,100</td>
</tr>
</tbody>
</table>

Table 2. The classes and attributes with definitions and scales for the organizational view of the modifiability metamodel.

described in section 3.

4.2.1. Defining probabilities for the causal structure

One way of assigning the conditional probabilities is to use expert elicitation methods such as surveys, interviews, or workshops [Keeney and Winterfeldt 1991]. However, it is both difficult and time consuming to get experts to assign probabilities directly into a PRM based metamodel. Most people are neither familiar with Bayesian network structures nor are they used to have discussions in terms of probabilities. Another common approach is to collect empirical data for the attributes and then use Bayesian network learning algorithms [Lauritzen 1995]. For example, measure the internal coupling, size, and understandability of a system in hundreds of cases, and then learn the probabilities of these attributes based on the collected data. This approach requires a lot of effort in order to collect the necessary amount of empirical data.
The available methods for learning Bayesian networks are too time consuming and do not consider expert opinions as a less time consuming option. Furthermore, no appropriate methods for expert elicitation have been found. The most common ones are also too time consuming since they either require experts familiar with Bayesian networks or an elaborative introductory phase. The proposed metamodel creation method employs a knowledge elicitation approach previously published in [Lagerström et al. 2009d]. This approach takes expert opinions into consideration when defining the conditional probability distributions without the need of introducing the experts to the concepts of conditional probabilities and PRMs.

The algorithms used for defining the CPD’s in the metamodel are based on the effect one attribute \( x \) have on a related attribute \( y \). In this case the effect has been found by a questionnaire among experts and is measured on an ordinal scale with three states High effect, Low effect, and No effect.

\[
P(y_j|x_i,n) = \begin{cases} 
\left( \frac{z_{1,n}}{z_{1,n} + z_{2,n} + z_{3,n}} \right) \frac{1}{z_{1,n} + z_{2,n} + z_{3,n}} & \text{if } i = j \\
\left( \frac{z_{2,n} + \frac{2}{3}z_{3,n}}{z_{1,n} + z_{2,n} + z_{3,n}} \right) \frac{1}{z_{1,n} + z_{2,n} + z_{3,n}} & \text{if } i \neq j \\
i \in \{1,2,3\} \ni j
\end{cases}
\]

Here \( n \) is an identification number of each causal dependency in the metamodel (for our running case regarding software change project cost cf. Fig. 6); \( i \) and \( j \) identifies the states, on a ordinal scale with three states, of the attributes \( x \) and \( y \) respectively. Finally \( z \) is the number of answers from the survey. In this case the representation is as following:

- \( z_{1,n} = \) number of High effect answers on question \( n \)
- \( z_{2,n} = \) number of Low effect answers on question \( n \)
- \( z_{3,n} = \) number of No effect answers on question \( n \)

When there is more than one attribute affecting the outcome then there will be a joint probability relation. A representation of the joint probability between attributes \( 1, \ldots, m \) is represented as \( P(Y|X_1, \ldots, X_m) \). For each alternative \( i \) with a corresponding output correlation \( j \) the joint probability is calculated as

\[
P(y_j|x_{i,1}, \ldots, X_{i,m}) = \sum_{n=1}^{m} \frac{P(y_j|x_{i,n})}{m}
\]

In the modifiability case followed in this paper, data for the CPD’s was collected in workshops and in a survey. The data collection is described in section 6.2 and the data presented in Table 7 serves as input for the causal structure CPD’s of the metamodel. The resulting CPD’s will look like the following examples.

Statement (causal dependency no 15 in Fig. 6 and Table 7): “How large is the effect of the systems external coupling on the need of synchronization between architectural change activities in a software change project?”

36 experts answered that the effect is high, 14 experts answered that the effect is low, and 1 expert said that there is no effect. With the presented expert elicitation method this translates to the CPD presented in Table 3.
A method for creating enterprise architecture metamodels – applied to systems modifiability analysis

<table>
<thead>
<tr>
<th>Systems external coupling (15)</th>
<th>Loose</th>
<th>Medium</th>
<th>Tight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architectural change activities</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>synchronization need</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td>0.9</td>
<td>0.05</td>
<td>0.05</td>
</tr>
<tr>
<td>Medium</td>
<td>0.05</td>
<td>0.9</td>
<td>0.05</td>
</tr>
<tr>
<td>High</td>
<td>0.05</td>
<td>0.05</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Table 3. The resulting conditional probability distribution for the attribute Architectural change activities synchronization need.

A joint probability example between four attributes and their joint effect on systems understandability is presented in Table 4. This corresponds to causal dependencies affecting the Systems understandability attribute in Fig. 6.

<table>
<thead>
<tr>
<th>Arch. doc. quality (10)</th>
<th>High</th>
<th>...</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>Systems internal coupl.</td>
<td>Loose</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(11)</td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Systems size (12)</td>
<td>Small</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Systems complexity (13)</td>
<td>Low</td>
<td>Med.</td>
<td>High</td>
</tr>
<tr>
<td>Systems understandability</td>
<td>Easy</td>
<td>0.92</td>
<td>0.68</td>
</tr>
<tr>
<td></td>
<td>Normal</td>
<td>0.04</td>
<td>0.28</td>
</tr>
<tr>
<td></td>
<td>Diff</td>
<td>0.04</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Table 4. The resulting conditional probability distribution for the attribute Systems understandability.

4.2.2. Defining the aggregate functions

As described in section 3 there are some attributes that are related as aggregates rather than by causality. In these cases, the probabilistic dependency is on some aggregate property over those attributes, such as the arithmetic operations **SUM** and **MEAN**.

In the metamodel, cf. Fig. 6, there are numerous classes having underlying classes that serve as Is-a-part-of classes. E.g. the change project class in the metamodel has a number of architectural and component change activity classes where the cost attribute aggregates by the **SUM** operation. While, in the organizational view, cf. Fig. 7, the architect team consists of a number of architects. Here, the expertise attribute of each architect aggregates by the **MEAN** operation to the expertise attribute of the architect team.

Since the aggregate functions are not based on causality but rather are aggregates of attributes, these CPD’s are not defined based on data and can not be found with experiments. These aggregate functions are definitions decided by the modeler and are in the modifiability metamodel case intuitively set as **SUM** or **MEAN**.
5. Modeling

Several case studies have been conducted based on the metamodel and enterprise architecture analysis approach presented. In all, 21 software change projects at 4 different companies have been analyzed with the modifiability metamodel so far.

One of the case studies was conducted at a large Nordic software and hardware vendor. Originally it consisted of two software change projects. In this paper, we detail one of them in order to describe how the created metamodel can be instantiated and used for cost analysis.

The company is a global leader in its domain. The company has over 100,000 employees worldwide and they have customers and local distributors in over 100 countries. The case study was carried out at one of the development departments at the company. The department’s work mainly focus on management and further development of an information and control system. This system is sold as a standard product, however each implementation is modified in order to match the unique requirements of the customer.

The projects studied in this case were two projects in a large program consisting of numerous projects. The projects, although part of a program, were treated by both the customer and the vendor as separate projects. The main program’s purpose was to implement a large upgrade of a National Control Center in a large non-European country. The upgrade consisted of a large amount of additional functionality that was supposed to be added to a standard product delivered by the vendor. In the project presented here, from now on called Project B, the main task was to add a new component to the standard product. The main function of this new component is to provide a tool for operators to evaluate and analyze optional transactions with other companies. The new component also required integration with several other already existing components of the system.

As discussed in Section 2 the modeling and assessment problems regarding correlation, causality, and uncertainties can and should be handled generally in the metamodel. However, the problems regarding measurement devices, scales, and discretization often need to be dealt with on a case-by-case basis, when the metamodel is put to use. For the case study presented here, the scales and measurement devices used for the organizational view attributes are presented in Table 5. As can be seen, most of the data was collected by interviewing the people in the project or by studying project documentation. Some of the continuous scales suggested in the metamodel were transformed to discrete scales in order to facilitate this kind of data collection.

Based on the data collection, an architecture model was instantiated for the project, as illustrated in Fig. 8. The instantiated model presents the project on a high level (viz. project view), but also highlights the architect and development team views with their aggregate objects. For the interested reader, the translation from probabilities to cost as number of man-hours is described in [Lagerström et al. 2009d].
Table 5. The classes and attributes with measurement devices and scales for the organizational view of the metamodel in the case study.

As can be seen in the organizational view of project B, cf. Fig. 8, there were six persons involved in project B. Five developers and one architect. Since many of the developers only had 0-2 years of experience for several of the experience attributes, the average expertise level of the developer team was assessed to be Low.

Project B was estimated to 800 man-hours by the project manager at the company based on his experience without any tool. As can be seen in Fig. 8, the instantiated model of project B provided an estimate of 3 085 man-hours. The project has now ended and the actual cost in man-hours was approximately 3 200.

Based on this one case the following points can be emphasized: 1) neither the expert nor the EA metamodel provided perfect estimations. 2) The expert estimate deviated 75 % from the actual value, while the metamodel provided an estimate deviating only 4 % from the actual value. 3) The metamodel enables early finding and calling to attention of risks. For instance, the instantiated model for project B shows that the developer team expertise was low and that the involved systems were complex.

A validation of the modifiability metamodel based on 10 software change projects has been discussed and published in [Lagerström et al. 2009d].

6. Validation

The qualitative structure of the metamodel proposed, i.e. the classes, reference slots, and attributes, needs to be evaluated and validated. While the use of academic
papers reflecting research serves as a good foundation for the metamodel creation, it is not completely trustworthy. There are several reasons for this. First, the scientific literature is not complete, so when creating a metamodel, it might be necessary to fill in some blanks with hypotheses unsupported by the literature. Second, the scientific literature is not always coherent, so when creating a metamodel, it might be necessary to make controversial choices. Third, the modeler might be biased and thus involuntarily introduce distortions. Expert validation of the metamodel serves as a good function in minimizing these uncertainties.

Validation data was gathered by presenting the metamodel to academic experts and practitioners, and having them discussing the content and the structure of the metamodel. The two most important questions are: 1) are there attributes missing in the metamodel that should be added? and 2) are there superfluous attributes in the metamodel that could be removed? Together, these questions determine whether the metamodel contains the appropriate attributes, and they will be addressed in subsections 6.1 and 6.2 respectively.

A third relevant question concerns the usability of the method as such, i.e. whether it is suitable for creation of metamodels for analysis of topics other than modifiability. This question is addressed in subsection 6.3.

A fourth question concerns the quantitative structure of the metamodel, i.e. the conditional probability distributions. Validation of these have been performed and...
published elsewhere [Lagerström et al. 2009b:d], so it will not be explicitly addressed in this paper.

6.1. Are there attributes missing in metamodel?

The first question, “are there attributes missing in the metamodel?”, was asked to a number of experts, both academic experts in an online survey and industrial experts during workshops. Two workshops with industrial experts, with 6 and 27 respondents respectively, were carried out. Of the 33 respondents 17 provided suggestions on attributes that could be missing in the metamodel. One online survey was provided to academics in the field of software maintainability, this survey had 40 respondents. Of the 40 respondents 13 provided suggestions on missing attributes. Thus, the total number of respondents were 73 and the total number of responses with suggestions on missing attributes were 30. The answers have been compiled into the list presented in Table 6.

<table>
<thead>
<tr>
<th>Class</th>
<th>Attribute</th>
<th>No of answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management</td>
<td>Degree of support</td>
<td>3</td>
</tr>
<tr>
<td>Testing process</td>
<td>Test coverage</td>
<td>2</td>
</tr>
<tr>
<td>System</td>
<td>Level of quality goals restrictions</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Platform independence</td>
<td>2</td>
</tr>
<tr>
<td>Requirements</td>
<td>Number of authors</td>
<td>3</td>
</tr>
<tr>
<td>specification</td>
<td>Number of changes during project</td>
<td>2</td>
</tr>
<tr>
<td>Architecture goals</td>
<td>Prioritized and communicated</td>
<td>3</td>
</tr>
<tr>
<td>Business organization</td>
<td>Stability</td>
<td>2</td>
</tr>
<tr>
<td>Change organization</td>
<td>Geographic distance (culture &amp; language diff.)</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Understandability of business objects</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Use of a common information model</td>
<td>3</td>
</tr>
<tr>
<td>Change activities</td>
<td>Time restrictions (deadline)</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 6. The workshop and survey results from the question “Which attributes are missing in the metamodel?”.

Since there is no strong agreement among the 30 experts on which attributes that are missing, the metamodel is considered to contain an appropriate set of attributes. The only suggestions that had more than one expert missing them are the attributes presented in Table 6. The attributes with the most votes in our workshops and survey had 3 persons out of the total 73 missing them, which is only 4.1%. If we consider the possibility that the 43 persons not missing any attributes at all are just convenient when answering the questionnaire, i.e. skips open-ended questions due to time restrictions, and instead restrict attention to the group that
did at all suggest new attributes we have 3 votes out of a total 30 (10%). Still, this is a rather small amount of agreement on which attributes that are missing in the metamodel. Nevertheless, the attributes suggested are interesting and should be explored in future case studies and surveys.

6.2. Does the metamodel contain attributes that can be removed?

The second question, "are there attributes in the metamodel that can be removed?", was addressed by analyzing the strength of the causal dependencies between the metamodel attributes, as extracted from the expert surveys. That is, if experts find that there is a strong causal connection between two attributes, than these attributes should be present in the metamodel. Conversely, if experts find that there is no causal connection between two attributes, than these attributes should be removed from the metamodel. Table 7 summarizes the causal dependencies between the attributes of the metamodel.

There were in total 99 experts participating. 13 are industrial experts providing answers at workshops and 86 are academic experts providing their answers via an online survey.

The question asked to the experts both in the workshops and the in survey was: “How large is the effect presented in the following statements?” Then the experts were provided with statements each corresponding to a causal dependency in the metamodel. See Fig. 6 for all the corresponding causal relationships. The statements were all arranged as the following examples; “Change management process maturity affecting architectural change activities cost”, which corresponds to the relationship labeled as number 1 in the metamodel. “Number of architects affecting the architectural change activities cost”, corresponding to the causal dependency labeled as number 2. The answers provided by the experts were given on the following scale; High effect, Low effect, No effect, and I don’t know.

In the workshops and the survey the persons were also provided with questions regarding their qualifications as experts. In the workshops, two persons were excluded due to lack of expertise; both persons stated that they had not enough experience in the field. Fourteen persons excluded from the survey either had too little experience, less than three years, they themselves said that they didn’t feel certain at all about their answers, or the answer I don’t know was given to more than 50% of the questions asked.

Since the attributes in general have either high effect or high/low effect in relation to their parents, whereas very few had no effect, the attributes in the metamodel all seem useful for modifiability analysis. As can be seen in Table 7, no causal dependency has more than 17.1% answers on No effect. We interpret these low percentages to mean that there are no attributes in the metamodel with no effect on its causally related attributes and by that the cost of making changes. Some causal dependencies in the metamodel do however have a number of answers on Low effect. These dependencies will be further validated in future change projects and surveys.
6.3. Method usability

In this paper, the proposed method for creating enterprise architecture metamodels for analysis has been exemplified with modifiability analysis, i.e. analysis of software change cost. However, the method is general and can be employed for different types of enterprise architecture analyses. The method has been used in a number of research projects resulting in a set of metamodels, each one tailored for a specific kind of analysis. The rest of this subsection presents this set of metamodels, in order to show the usability of the proposed metamodel creation method.

In [Närman et al. 2007] a metamodel for system quality analysis called PERDAF is presented. The PERDAF metamodel was developed in order to support analysis of a set of system quality attributes, viz. maintainability, security, reliability, usability, efficiency, interoperability, suitability, and accuracy.

Ullberg et al. focus their work on interoperability analysis with enterprise architecture models. [Ullberg et al. 2008] presents a metamodel that supports the creation of enterprise architecture models amenable to analysis of enterprise service interoperability.

Dependency analysis with fault trees is another application of enterprise architecture metamodels. In [Franke et al. 2009a] Franke et al. describes how a method similar to ours can be used to tailor the DoDAF enterprise architecture framework.

Table 7. Survey and workshop data regarding the strength of influence between causally related attributes in the metamodel.

Possibly one or two attributes can be removed or replaced but this requires more research.
so that it supports probabilistic dependency analysis of the kind described in this paper.

König and Nordström presents a metamodel for impact analysis of system quality on the operation of active distribution grids [König and Nordström 2009]. In this paper the work is focused on runtime ICT qualities such as accuracy, performance, availability, and security.

In [Höök et al. 2009] an enterprise architecture metamodel for ICT system impact on maintenance management is presented. The metamodel supports the maintenance management for electric power utilities and Höök et al. employs our proposed method when tailoring the general ArchiMate-metamodel into a metamodel specific for maintenance management.

[Gustafsson et al. 2009] presents a metamodel for business value analysis. The metamodel focuses on analyzing a number of business values, viz. flexibility, efficiency, integration, coordination, decision making, control and follow up, and organizational structure.

7. Related work

This section will provide some related work with respect to metamodel creation methods.

A number of EA initiatives have been proposed, e.g. The Open Group Architecture Framework (TOGAF) [The Open Group 2008], the Zachman Framework [Zachman 1987], and Enterprise Architecture Planning (EAP) [Spewak and Hill 1992]. There are also numerous metamodels proposed for EA modeling, e.g. the ones presented by O’Rourke [O’Rourke et al. 2003] and Lankhorst [Lankhorst 2005]. These initiatives tend to focus on the differentiation of metamodels in viewpoints, often referred to as architecture layers. Neither the larger frameworks nor the metamodel oriented initiatives typically contain methods for metamodel design or adaptation to suit specific analyses (except on a very general level). Only a few metamodels detail whether and how they support decision making.

In contrast to the general and enterprise-wide modeling languages found within the discipline of EA, there exist a large number of languages that serve more specific purposes. Software and system architecture description languages, for instance, focus on internal structure and design of software systems. In addition to capturing the overall structure of systems, analysis capabilities are often available, such as the availability, security, and timeliness analyses in the Architecture Analysis and Design Language (AADL) [Society of Automotive Engineers 2009]. Other examples of modeling languages with very specific purposes are found, for instance, in software security engineering, where languages such as UMLsec [Jürjens 2005] and secure UML [Lodderstedt et al. 2002] have been tailored specifically for security analysis. These languages provide good support for detailed modeling for one specific type of analysis. However, they lack holistic scope, which means that a subject such as security is only covered from a limited (typically technical) point of
A method for creating enterprise architecture metamodels – applied to systems modifiability analysis

view. Furthermore, they also miss a large class of analyses such as system modifiability, business/IT alignment, and interoperability that are highly relevant when considering an enterprise as a whole.

Looking even broader we quickly turn to adjacent areas. There is much written on the subject of knowledge elicitation [Gaines and Shaw 1994], i.e. how to capture existing knowledge about phenomena in a systematic way. This field of research is associated with artificial intelligence [Liou 1990], requirements elicitation [Nuseibeh et al. 1994], ontological engineering [Corcho et al. 2004], and Bayesian networks [Kadane and Wolfson 1998]. Most of the methods within these research fields focus on conducting experiments or interviews. Most of the methods are customized for a specific purpose, and since none of the methods mentioned deal with metamodeling it is difficult to employ them as metamodel design methods off-the-shelf.

To conclude, frameworks and formal notations for EA are available. However, general and notation independent methods for metamodeling that support decision making and analysis are rare. None of the methods, frameworks, and notations examined results in metamodels, covers the entire EA domain, and is analysis oriented. Furthermore, there is little support of causality formalization or attributes in the available methods for enterprise metamodel creation.

8. Conclusions

This paper proposes a structured method for creating enterprise architecture metamodels amenable to analysis. The method focuses on a causality based goal breakdown that produces metamodel fragments with well defined elements, controllable by the decision maker. The method contains guidelines for knowledge elicitation based on scientific sources during the metamodel creation phase. Once the goal break-down fragments have been created, these can be translated into metamodel fragments. Guidelines are proposed for how to merge these fragments into a single metamodel.

Enterprise systems modifiability, i.e. software change cost, is used as a running case illustrating the method. The modifiability metamodel is presented and instantiated based on a case study at a large Nordic software and hardware vendor, showing the applicability of the produced metamodel.

Furthermore, the paper discusses the validity of the metamodel creation method based on data collected in workshops and surveys with both academia and industry. This data shows that the proposed metamodel for modifiability contains the appropriate elements. We also discussed the usability of the method by presenting a number of cases in which the method has been used and where it has produced a set of metamodels for different types of analyses. Thus, the applicability of the method beyond the modifiability example in the present paper has been demonstrated.
References


Jensen, F. V. (2001). *Bayesian Networks and Decision Graphs*. Springer-Verlag
New York, Inc., Secaucus, NJ, USA.


Lauritzen, S. (1995). The em algorithm for graphical association models with miss-
REFERENCES


Chapter 5

Architecture Analysis of Enterprise Systems Modifiability
A Metamodel for Software Change Cost Estimation

Robert Lagerström · Pontus Johnson · Mathias Ekstedt

Abstract  Enterprise architecture models can be used in order to increase the general understanding of enterprise systems and specifically to perform various kinds of analysis. The present paper proposes a metamodel for enterprise systems modifiability analysis, i.e. assessing the cost of making changes to enterprise-wide systems. The enterprise architecture metamodel is formalized using probabilistic relational models, which enables the combination of regular entity-relationship modeling aspects with means to perform enterprise architecture analysis. The content of the presented metamodel is validated based on survey and workshop data and its estimation capability is tested with data from 21 software change projects. To illustrate the applicability of the metamodel an instantiated architectural model based on a software change project conducted at a large Nordic transportation company is detailed.

Keywords  Enterprise architecture · Software modifiability · Metamodel · Probabilistic relational models

1 Introduction

Managing software systems today is a complex business (Ross et al. 2006). In order to achieve effective and efficient management of the software system landscape it is essential to be able to assess the current status of system qualities such as availability, performance, security, and modifiability, as well as estimate their values in different future scenarios (Bass et al. 1998). Estimation of these qualities is however a great challenge that to a large extent can be addressed by introducing relevant models as a means of abstraction, which can
be achieved with enterprise architecture modeling (Johnson and Ekstedt 2007). This paper presents an enterprise architecture metamodel that supports analysis of systems modifiability.

1.1 Enterprise architecture

In recent years, Enterprise Architecture (EA) has become an established discipline for business and software system management (Ross et al. 2006). EA describes the fundamental artifacts of business and IT as well as their interrelationships (Zachman 1987; Lankhorst 2005; Ross et al. 2006; Winter and Fischer 2007; The Open Group 2009). Architecture models constitute the core of the approach and serve the purpose of making the complexities of the real world understandable and manageable to humans (Winter and Fischer 2007). A main concept in EA is the metamodel which acts as a pattern for the instantiation of the architectural models. In other words, a metamodel is a description language used when creating models (Lankhorst 2005; Johnson and Ekstedt 2007; Kurpjuweit and Winter 2007; The Open Group 2009). EA ideally aids the stakeholders of the enterprise to effectively plan, design, document, and communicate IT and business related issues, i.e. they provide decision support for the stakeholders (Kurpjuweit and Winter 2007).

A related discipline is software architecture/IT-system architecture. (Bass et al. 1998) defines software (IT-system) architecture of a program or computing system as the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them. In the software architecture discipline the architecture concept is limited to include components, their properties and the relations within a software system. However, large contemporary enterprises do not have one or even a few software systems to manage. They have hundreds or even thousands of systems. Also, there are only a small amount of standalone applications in the modern software system landscape. Many systems are tightly integrated with each other. Furthermore, the management of software systems is no longer an isolated task of one or few engineers. The complex changes being implemented involve business executives, project managers, architects, developers, testers etc. Therefore, when considering software system management issues for large contemporary enterprizes software architecture alone will not be sufficient. However, enterprise architecture modeling is appropriate since it considers the software system architecture with components and relations and the documentation, people, processes etc.

In relation to providing decision-making support, a key underlying assumption of the EA models is that they should provide some more aggregated knowledge than what was merely put into the model in the first place. Software application architecture, for instance, does not only keep track of the set of systems in an enterprise and their internal relationships, it also provides information about the dependencies between the systems. More broadly, the dependencies between the business and the software systems are covered in an EA. So, conclusions can for instance be drawn about the consequences in the enterprise if one specific system became unavailable.

This type of analysis of EA can benefit for its stakeholders. Unfortunately however, EA frameworks rarely explicitly state either what kind of analyses can be performed given a certain model or how the analysis should be performed in detail (Johnson et al. 2007; Franke et al. 2009b). This paper uses a formalized approach to enable analysis of EA models. The underlying fundamental formalism of the approach is called Probabilistic Relational
Models (PRMs) (Friedman et al. 1999), which in turn employs the statistical mathematics of Bayesian networks (Neapolitan 2003; Jensen 2001).

1.2 Enterprise system modifiability

As discussed in the previous subsection, enterprise architecture models can be used to analyze different system qualities and provide information for the decision maker regarding different scenarios. In this paper the focus will be on enterprise software system modifiability, i.e. the cost of making changes to enterprise-wide software systems.

Business environments today progress and change rapidly to keep up with evolving markets. Most business processes are supported by software systems and as the business processes change, the systems need to be modified in order to continue supporting the processes. Modifications include extending, deleting, adapting, and restructuring the enterprise systems (Bass et al. 1998). The modification effort ranges from adding a functional requirement in a single system to implementing a service-oriented architecture for the whole enterprise.

An essential issue with today’s software systems is that many of them are interconnected, thus a modification to one system may cause a ripple effect among other systems. Also, numerous systems have been developed and modified over many years. Making further changes to these systems might require a lot of effort from the organization, for example due to a large number of previous modifications implemented ad hoc. Problems like these raise questions for IT-decision makers such as: Is there enough documentation describing the systems, and has the documentation been updated correctly after each modification? Is the source code easy to understand? Which systems are interconnected?

Several studies show that the modification work is the phase of a system’s lifecycle that consumes the greatest portion of resources; (Harrison and Cook 1990) report that over 70 % of the software budget is spent on maintenance, (Pigoski 1997) refers to studies stating that the maintenance cost, relative to the total life cycle cost of a software system, has been increasing from 40 % in the early 1970s up to 90 % in the early 1990s, and (Jarzabek 2007) states that “the cost of maintenance, rather than dropping, is on the increase”.

The activities of modifying enterprise systems are typically executed in projects, and IT-decision makers often find it difficult to estimate and plan their change projects. Thus, a large proportion of the projects aiming to modify a software system environment fail. That is, the projects tend to take longer time and cost more than expected. (Laird and Brennan 2006) state that 23 % of the software projects are cancelled before completion, whereas of those completed only 28 % were delivered on time, and the average software project overran the budget by 45 %. This can often occur due to lack of information about the systems being changed. According to (Laird and Brennan 2006), software engineers must be able to understand and predict the activities, as well as manage the risks, through estimation and measurement. Therefore, it would be useful for IT-decision makers to gather more information in a structured manner and use this information to analyze how much effort a certain modification to an enterprise software system would require.

This paper will address these issues of software change by employing enterprise architecture modeling for systems modifiability analysis, thus providing a metamodel for assessment of software change project cost.
1.3 Outline

The remainder of the paper is structured as follows: In section 2 related work is presented. Section 3 presents the probabilistic relational models which serve as the underlying formalism for the enterprise architecture metamodel proposed for analysis. The following section goes through the enterprise architecture metamodel creation method used when designing the modifiability metamodel. Next, in section 5 the modifiability metamodel is thoroughly described. Section 6 contains data for validation of the proposed metamodel by considering the correctness of the qualitative structure and the estimation capabilities of the quantitative structure. In section 7 the estimation capabilities of the metamodel are compared with the estimation capabilities of other models and methods available. An instantiated architectural model for a software change project is described in section 8 in order to illustrate the applicability of the presented metamodel. Section 9, discusses the research and future work. Finally, section 10 summarizes the paper with conclusions.

2 Related work

When developing the metamodel for system modifiability analysis using enterprise architecture models, two research disciplines were mainly covered: software system modifiability and enterprise architecture. This section presents the related work within these two disciplines.

2.1 Modifiability analysis

The issue of dealing with modifiability is not an enterprise architecture specific problem. Managing and assessing system change has been addressed in research for many years. Some of the more well-known assessment approaches include the CONstructive COst MOdel (COCOMO), the Software Architecture Analysis Method (SAAM), the Oman taxonomy, and the ISO/IEC 9126 standard. These are briefly described below.

COCOMO, CONstructive COst MOdel, was in its first version released in the early 1980's. It became one of the most frequently used and most appreciated software cost estimation models of that time. Since then, development and modifications of COCOMO have been performed several times to keep the model up to date with the continuously evolving software development trends. The latest version of COCOMO, called COCOMO II, had its estimation capabilities calibrated in the year 2000 with the help of information from 161 project data points and 8 experts. This latest calibrated version of COCOMO II uses the probabilistic Bayesian approach for turning a priori obtainable data into estimates of costs related to an a posteriori state of a software development or modification project (Boehm 1981; Chulani et al. 1999; Boehm et al. 2000).

(Bass et al. 1998) proposes the Software Architecture Analysis Method (SAAM) for software quality evaluation. This method takes several quality attributes into consideration; performance, security, availability, functionality, usability, portability, reusability, testability, integrability and modifiability. Bass et al. categorizes modifications as: extending or changing capabilities, deleting unwanted capabilities, adapting to new operating environments and restructuring. Based on the quality attributes presented, Bass et al. propose different architectural styles which then are employed in the SAAM. SAAM is a scenario-based approach
which intends to make sure that stakeholder quality goals are met (for instance high modifi-
ability). According to (Bass et al. 1998) SAAM can be used in two contexts: as a validation
step for an architecture being developed or as a step in the acquisition of a software system.
Besides SAAM there are several other methods supporting analysis of software architecture
quality attributes, such as Architecture Trade-off Analysis Method (ATAM) (Kazman et al.
2000), Cost Benefit Analysis Method (CBAM) (Kazman et al. 2001), Architecture Level
Modifiability Analysis (ALMA) (Bengtsson 2002), and Aspectual Software Architecture
Analysis Method (ASAAM) (Tekinerdogan 2004).

The Definition and Taxonomy for Software Maintainability presented in (Oman et al.
1992) provides a hierarchical definition of software maintainability in the form of a taxon-
omy. (Oman et al. 1992) found three broad categories of factors influencing the maintain-
ability of a software system; management, operational environment, and the target software
system. Each of these top-level categories is then further broken down into measurable at-
tributes. According to (Oman et al. 1992) the taxonomy can be useful for developers by
defining characteristics affecting the software maintenance cost of the software they are de-
veloping. Hence, the developers can write highly maintainable software from the beginning
by studying the taxonomy. Maintenance personnel can use the taxonomy to evaluate the
maintainability of the software they are working with in order to pin point risks etc. Project
managers and architects can use the taxonomy in order to prioritize projects and locate areas
in need of re-design.

ISO/IEC 9126 (International Organization for Standardization 2001; 2003a;b) is an in-
ternational standard for software engineering focusing on software quality. The proposed
quality model contains six quality attributes; functionality, reliability, usability, efficiency,
maintainability, and portability. The aim with this quality model is to provide definitions
of the quality attributes and provide a set of sub-characteristics that influence these quality
attributes. ISO/IEC defines maintainability as; “the capability of the software product to be
modified. Modifications may include corrections, improvements or adaptation of the soft-
ware to changes in environment, and in requirements and functional specifications”. Main-
tainability is divided into analyzability, changeability, stability, and testability. For each of
these sub-characteristics ISO/IEC provides a set of metrics for evaluation. According to
ISO/IEC the quality models can be used to validate the completeness of a requirements defi-
nition, identify software requirements, identify software design objectives, identify software
testing objectives, identify quality assurance criteria, and identify acceptance criteria for a
completed software product.

The available methods for modifiability analysis are not focusing on change in an enter-
prise architecture context. There are many problems that need to be addressed that the avail-
able methods miss, such as: the increasing number of systems affected by enterprise-wide
changes, the tight integration between systems, the increasing involvement of diverse peo-
ples in a company e.g. business executives, project managers, architects, developers, testers.
Some methods do use models, other employ quality criteria, some has a formal analysis en-
gine, and there are methods using scenarios in decision making situations. There is however
no method having brought it all together in an EA context. The studied methods provided
valuable input for the EA approach presented in this paper and serve as the main references
for the modifiability metamodel.
2.2 Enterprise architecture

As stated in subsection 1.1, the exact procedure or algorithm for how to perform a certain analysis given an architecture model is very seldom provided by EA frameworks. Most frameworks do however recognize the need to provide special purpose models and provide different viewpoints intended for different stakeholders. Unfortunately however, most viewpoints are designed from a model entity point of view, rather than a stakeholder concern point of view. Thus, assessing a quality such as the modifiability of a system is not something that is performed in a straightforward manner. The Department of Defense Architecture Framework (DoDAF) (Department of Defense Architecture Framework Working Group 2007) for instance, provides products (i.e. viewpoints) such as "systems communications description", "systems data exchange matrix", and "operational activity model". These are all viewpoints based on a delimitation of elements of a complete metamodel, and they are not explicitly connected to a certain stakeholder or purpose. The Zachman framework presented in (Zachman 1987; 2009), does connect model types describing different aspects (Data, Function, Network, People, Time, and Motivation) with very abstractly described stakeholders (Strategists, Executive Leaders, Architects, Engineers, and Technicians), but does not provide any deeper insight how different models should be used. The Open Group Architecture Framework (TOGAF) (The Open Group 2009), explicitly states stakeholders and concerns for each viewpoint they are suggesting. However, neither the exact metamodel nor the mechanism for analyzing the stated concerns, are described. In relation to modifiability, the most appropriate viewpoints provided would, according to TOGAF, arguably be the Software Engineering View, the Systems Engineering View, the Communications Engineering View, and the Enterprise Manageability View. In the descriptions of these views one can find statements such as; "the use of standard and self-describing languages, e.g. XML, are good in order to achieve easy to maintain interface descriptions". However, the exact interpretation of such statements when it comes to architectural models or how it relates to the modifiability of a system as a whole, is left out. Moreover, these kinds of "micro theories" are only exemplary and do not claim to provide a complete theory for modifiability or similar concerns.

Other, more formalized analysis mechanisms for enterprise architecture models may be found in e.g. (Lankhorst 2005) for performance and availability, for software architecture languages in (Allen 1997) where for instance dead-lock and interoperability analyses are provided. Architecture Analysis and Design Language (AADL) (Society of Automotive Engineers 2009) provides availability, security, and timeliness analyses, and UMLsec (Jürjens 2005) provides security analysis. None, however, offer architecture models for software system modifiability analysis.

Another permeating problem in EA modeling is the uncertainty that is related to the model. For instance, whether a model is the result of a very thorough and recent investigation or a quick read-through of old documents will impact the quality of the decision support the model constitutes for its various stakeholders. (Johansson 2005) propose a method to handle this kind of modeling uncertainty.

Besides the data collection uncertainties, there are also uncertainties related to the elements in the models. For instance, are all the software systems in the model still in use, is the data flow still as depicted and is the process structure really illustrating what is really happening? This kind of uncertainty is not addressed by today’s EA frameworks. The users of a model are simply left to their best knowledge or gut feeling when estimating to what extent the EA model, and the analyses it is subjected to, can and should be trusted. (Aier et al. 2009) address these issues by "demonstrating the feasibility of applying the life table
method to assess life spans of EA artifacts via calculating the probability of particular applications to survive a certain number of years", i.e. they show that there is an uncertainty related to EA artifacts. Since, in an enterprise, these artifacts change over time so must the models. However, (Aier et al. 2009) do not explain how this is taken care of explicitly in EA models.

Other EA initiatives using the probabilistic relational models formalism, or similar, for analysis are available. In (Närman et al. 2007) a metamodel for system quality analysis called PERDAF is presented. The PERDAF metamodel was developed in order to support analysis of a set of system quality attributes, viz. security, reliability, usability, efficiency, interoperability, suitability, and accuracy. (Ullberg et al. 2008) presents a metamodel that supports the creation of enterprise architecture models amenable to analysis of enterprise service interoperability. Dependency analysis with fault trees is another application of enterprise architecture metamodels. (Franke et al. 2009a) describes a method that can be used to tailor the DoDAF enterprise architecture framework so that is supports probabilistic dependency analysis. (König and Nordström 2009) presents a metamodel for impact analysis of system quality on the operation of active distribution grids. In this paper the work is focused on runtime ICT-qualities such as accuracy, performance, availability, and security. In (Höök et al. 2009) an enterprise architecture metamodel for ICT system impact on maintenance management is presented. The metamodel supports the maintenance management for electric power utilities. (Gustafsson et al. 2009) presents a metamodel for business value analysis. The metamodel focuses on analyzing a number of business values, viz. flexibility, efficiency, integration, coordination, decision making, control and follow up, and organizational structure.

Since there are no EA frameworks or metamodels focusing on modifiability analysis available, the present paper aims at filling this gap in enterprise architecture. The approach also copes with uncertainties by not considering the information in EA models as fixed constants but rather as probabilities. As stated in the introduction this paper uses a formalized approach to enable analysis of EA models under uncertainty called probabilistic relational models (PRMs).

3 Probabilistic relational models

As stated in the introduction Probabilistic Relational Models (PRMs) serve as the underlying formalism for the enterprise architecture metamodel proposed for modifiability analysis. Previously proposed formalisms are presented in (Johnson et al. 2007; Lagerström 2007).

A PRM (Friedman et al. 1999) specifies a template for a probability distribution over an architecture model. The template describes the metamodel for the architecture model, and the probabilistic dependencies between attributes of the architecture objects. A PRM, together with an instantiated architecture model of specific objects and relations, defines a probability distribution over the attributes of the objects. The probability distribution can be used to infer the values of unknown attributes, given evidence of the values of a set of known attributes.

An architecture metamodel $\mathscr{M}$ describes a set of classes, $\mathcal{X} = X_1, \ldots, X_n$. Each class is associated with a set of descriptive attributes and a set of reference slots. The set of descriptive attributes of a class $X$ is denoted $\mathcal{A}(X)$. Attribute $A$ of class $X$ is denoted $X.A$ and its domain of values is denoted $V(X.A)$. For example, a class System might have the descriptive attribute $Size$, with domain $\{large, medium, small\}$. The set of reference slots of a class $X$ is denoted $\mathcal{R}(X)$. We use $X.\rho$ to denote the reference slot $\rho$ of $X$. Each reference slot $\rho$ is
typed with the domain type $\text{Dom}[\rho] = X$ and the range type $\text{Range}[\rho] = Y$, where $X; Y \in \mathcal{X}$. A slot $\rho$ denotes a relation from $X$ to $Y$ in a similar way as Entity-Relationship diagrams. For example, we might have a class `Documentation` with the reference slot `Describes` whose range is the class `System`.

An architecture instantiation $\mathcal{I}$ (or an architecture model) specifies the set of objects in each class $X$, the values for the attributes, and the reference slots of the objects. For example, Fig. 8 presents an instantiation of the change project metamodel of Fig. 7. It specifies a particular set of changes, systems, documents, etc., along with values for each of their attributes and references. For future use, we also define a relational skeleton $\sigma_i$ as a partial instantiation which specifies the set of objects in all classes as well as all the reference slot values, but not the attribute values.

A probabilistic relational model $\Pi$ specifies a probability distribution over all instantiations $\mathcal{I}$ of the metamodel $\mathcal{M}$. This probability distribution is specified as a Bayesian network (Jensen 2001), which consists of a qualitative dependency structure and associated quantitative parameters.

The qualitative dependency structure is defined by associating with each attribute $X.A$ a set of parents $\text{Pa}(X.A)$. Each parent of $X.A$ has the form $X.\tau.B$ where $B \in \mathcal{M}(X.\tau)$ and $\tau$ is either empty, a single slot $\rho$ or a sequence of slots $\rho_1, \ldots, \rho_k$ such that for all $i$, $\text{Range}[\rho_i] = \text{Dom}[\rho_{i+1}]$. For example, the attribute `Cost` of class `ChangeProject` may have `ChangeOrganization.Developer.Expertise` as parent, thus indicating that the cost of a prospective software modification project depends on the expertise of the developers employed in the organization. Note that $X.\tau.B$ may reference a set of attributes rather than a single one. In these cases, we let $x.A$ depend probabilistically on some aggregate property over those attributes, such as the logical operations AND, OR, and NOR. In this paper we use the arithmetic operations SUM and MEAN as aggregate functions. For instance, if there are several developers engaged in a modification project, we might aggregate the individual developers’ expertise into a mean expertise of the whole development team.

Considering the quantitative part of the PRM, given a set of parents for an attribute, we can define a local probability model by associating a conditional probability distribution (CPD) with the attribute, $P(X.A|\text{Pa}(X.A))$. For instance, $P(\text{ChangeProject.Cost} = \text{high}|\text{ChangeOrganization.Developer.Expertise} = \text{low}) = 90\%$ specifies the probability that the project cost will be high, given the expertise of the developers involved.

We can now define a PRM $\Pi$ for a metamodel $\mathcal{M}$ as follows. For each class $X \in \mathcal{X}$ and each descriptive attribute $A \in \mathcal{M}(X)$, we have a set of parents $\text{Pa}(X.A)$, and a CPD that represents $P_{\Pi}(X.A|\text{Pa}(X.A))$.

Given a relational skeleton $\sigma_i$ (i.e. a metamodel instantiated to all but the attribute values), a PRM $\Pi$ specifies a probability distribution over a set of instantiations $\mathcal{I}$ consistent with $\sigma_i$:

$$P(\mathcal{I} | \sigma_i, \Pi) = \prod_{x \in \sigma_i(X)} \prod_{A \in \mathcal{M}(x)} P(x.A|\text{Pa}(x.A))$$

where $\sigma_i(X)$ are the objects of each class as specified by the relational skeleton $\sigma_i$.

A PRM thus constitutes a formal machinery for calculating the probabilities of various architecture instantiations. This allows us to infer the probability that a certain attribute assumes a specific value, given some (possibly incomplete) evidence of the rest of the architecture instantiation. In addition to express and infer uncertainty about attribute values as specified above, PRMs also provide support for specifying uncertainty about the structure of the instantiations. A more detailed description of this topic is, however, beyond the scope of the paper.
4 Creating the modifiability metamodel

Creating a good metamodel (probabilistic relational model) is not trivial. Obviously, it is important that the metamodel is tailored for the management tasks it should support, i.e. the kind of analysis the metamodel is intended to perform. For instance, if one seeks to employ an enterprise architecture model for evaluating business process efficiency, the information required from the model differs radically from the case when the model is used to evaluate the modifiability of an enterprise software system. This section aims at presenting the method employed when the modifiability metamodel was designed. For the interested reader a more detailed description of the method is presented in (Lagerström et al. 2009a).

4.1 Method for creating the qualitative part of the metamodel

This subsection presents the method for creating the qualitative part of the modifiability metamodel, i.e. the classes, reference slots, attributes and their parents.

The method for creating decision support metamodels focus on finding a set of appropriate a priori measures for a chosen goal, i.e. finding measures with high correlation and causal influence on the selected goal. This can be done in several ways, for instance by studying research literature, doing experiments and case studies, or using expert opinions.

The first step in the method is to select the goal that the metamodel under design is supposed to support. In this case the goal considered is modifiability, i.e. change cost. Then, variables causally affecting the goal and the variables found in previous iterations are identified. This iterative process continues until all paths of variables, and causal relations between them, have been broken down into variables that are directly controllable by the decision maker, see part 1 and 2 of Fig. 1. The iterative process of finding variables affecting modifiability is supported by knowledge elicitation guidelines and control steps. These have been described in (Lagerström et al. 2009a; 2007; Johnson et al. 2007; Lagerström et al. 2009d). The result of the iterative process is a set of goal break-down fragments. Each fragment is based on scientific knowledge, and exhibits variables that are all causally linked to a well-defined goal, controllable by the decision maker, e.g. as in part 3 of Fig. 1. The next step is to translate these into metamodel classes, attributes, and reference slots. The attributes of the metamodel classes correspond to the variables found in the goal break-down part of the method. The goal break-down fragment presented in part 3 of Fig. 1 corresponds to the metamodel fragment presented in part 4 of the same figure.

Fig. 1 The iterative goal break-down process, a resulting goal break-down fragment example, and the metamodel fragment related to it.
The method steps described so far result in a number of metamodel fragments, all based on a selected source or set of sources. However, depending on the granularity of the sources, these fragments are usually very small and local models, i.e. models describing the relations between a few found elements in great detail, but without a sense of a bigger picture. Furthermore, the fragments are sometimes completely disjoint, sometimes completely overlapping, and usually somewhere in between – all depending on the scope of the original sources. To make full use of the knowledge elicited a merge of the fragments into one metamodel is needed. The challenge is to make sure that the metamodel remains coherent and non-ambiguous, despite its diverse origins. This integration challenge is addressed in (Lagerström et al. 2009a; 2008), which provides some guidelines for merging.

By using the guidelines for metamodel fragment merge, a metamodel was finally obtained for modifiability analysis. Fig. 2 depicts the classes and reference slots. Fig. 3 – Fig. 6 present different views of the modifiability metamodel including the aggregating functions and classes. Fig. 7 contains the classes, attributes, and causal structure of the metamodel, excluding the aggregating classes.

4.2 Method for creating the quantitative part of the metamodel

The former subsection presented the method for creating the qualitative part of the modifiability metamodel. This subsection presents the second part of the method, creating the quantitative part of such a metamodel. That is, to define the conditional probability distributions (CPDs) related to each attribute.

The proposed metamodel creation method employs a knowledge elicitation approach previously published in (Lagerström et al. 2009a;c). This approach takes expert opinions into consideration when defining the conditional probability distributions without the need of introducing the experts to the concepts of conditional probabilities and PRMs. The algorithms used for defining the CPDs in the metamodel are based on the effect one attribute $x$ has on a related attribute $y$. In the modifiability case the effect was found by using a questionnaire among experts and is measured on an ordinal scale with three states High effect, Low effect, and No effect.

$$
P(y_j|x_i,n) = \begin{cases} 
\frac{z_{1,n} + \frac{2}{3}z_{2,n} + \frac{1}{3}z_{3,n}}{z_{1,n} + z_{2,n} + z_{3,n}} & \text{if } i = j \\
\frac{1}{3}z_{2,n} + \frac{2}{3}z_{3,n} & \text{if } i \neq j 
\end{cases}$$

$i \in \{1, 2, 3\} \Rightarrow j$

Here $n$ is an identification number of each causal dependency in the metamodel (cf. Fig. 7); $i$ and $j$ identify the states, on an ordinal scale with three states, of the attributes $x$ and $y$ respectively. Finally $z$ is the number of answers from the survey. In this case the representation is as following:

$z_{1,n} =$ number of High effect answers on question $n$

$z_{2,n} =$ number of Low effect answers on question $n$

$z_{3,n} =$ number of No effect answers on question $n$

When there is more than one attribute affecting the outcome then there will be a joint probability relation. A representation of the joint probability between attributes $1, \ldots, m$ is represented as $P(Y|X_1, \ldots, X_m)$. For each alternative $i$ with a corresponding output correlation $j$ the joint probability is calculated as

$$
P(y_j|x_{i_1}, \ldots, x_{i_m}) = \sum_{n=1}^{m} \frac{P(y_j|x_{i,n})}{m}$$
In the modifiability case, data for the CPDs was collected in workshops and surveys. The data collection is described in section 6.2 and the data presented in Table 3 serves as input for the causal structure CPDs of the metamodel.

A joint probability example between four attributes and their joint effect on components change difficulty is presented in Table 1. This corresponds to causal dependencies affecting the Change difficulty attribute of the class Technical changes to components in Fig. 7.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>High</th>
<th></th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comp. change env. tool quality (24)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comp. change env. infra quality (23)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tech. change to comp. change size (23)</td>
<td>Low</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Components understandability (22)</td>
<td>Easy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tech. change to comp. change diff.</td>
<td>Normal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tech. change to comp. change diff.</td>
<td>Difficult</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1 The resulting conditional probability distribution for the attribute Change difficulty of the class Technical changes to components.

As described in section 3 there are some attributes that are related as aggregates rather than by causality. In these cases, the probabilistic dependency is on some aggregate property over those attributes, such as the arithmetic operations SUM and MEAN. In the metamodel, cf. Fig. 2 – Fig. 7, there are numerous classes having underlying classes that serve as Is-a-part-of classes. E.g. the change project class in the metamodel has a number of architectural and component change activity classes where the cost attribute aggregates by the SUM operation, cf. Fig. 3. While, in the documentational view, cf. Fig. 5, the architectural documentation consists of a number of documents, here the quality attribute of each document aggregates by the MEAN operation to the quality attribute of the architectural documentation. Since the aggregate functions are not based on causality but rather on aggregates of attributes, these CPDs are not defined based on data and can not be found with experiments. These aggregate functions are definitions decided by the modeler and in the modifiability metamodel case are intuitively set as SUM or MEAN.

5 The modifiability metamodel

This section presents the enterprise architecture metamodel for modifiability analysis. In subsection 5.1 the classes and reference slots of the metamodel are presented and in subsection 5.2 the focus is on describing the attributes and how these are related.

5.1 Classes and reference slots

The proposed metamodel for modifiability analysis, cf. Fig. 2, focuses on the software systems and the surrounding environment involved in or affected by the modifications implemented in a change project, thus aiming at analyzing modifiability defined as change project cost (high modifiability leads to low change costs). The main metamodel element is therefore the Change project class. Modifications carried through in change projects may include extending, deleting, adapting, and restructuring the enterprise systems (Bass et al. 1998).
This could for instance be increasing a non-functional requirement, adding a new function in a system, or integrating two systems.

Change projects are divided into Architectural change activities and Component change activities. Architectural change activities are the activities concerning modifications on an architecture level, i.e. involving several systems or components, while component change activities concern modifications to a single component of an application.

The change activities perform Technical changes to the architecture and Technical changes to the components, thus the metamodel is supposed to be used for modeling the actual changes for each activity.

Modifications are implemented in either Systems or Components. A system is a collection of components organized to accomplish a specific function or a set of functions (IEEE Standards Board 1990). (IEEE Standards Board 1990) defines a component as: "One of the parts that make up a system. A component may be hardware or software and may be subdivided into other components. The terms module, component, and unit are often used interchangeably or defined to be sub-elements of one another in different ways depending on the context. The relationship of these terms is not yet standardized.” The presented metamodel does not differentiate these concepts either and will use the term component.

According to (The IT Governance Institute 2007) all modifications related to software systems need to be formally managed in a controlled manner, this includes changes to be logged, assessed and authorized prior to implementation. For change projects the Change management process intends to support this. There are four important parts of this process; the activities that define the process, the roles assigned to these activities, the documents that serve as input and output in the process, and the metrics used to assess and control the process.

The Change organization refers to the organizations designing and implementing the software system modifications, i.e., the parties involved in the architectural and component changes, such as consultants, application vendors, in-house resources etc.

The change organization contains Architects and Developers. Architects are the people in the change project who design and modify the architecture of the enterprise systems. Developers, on the other hand are the ones writing and modifying the source code of the enterprise architecture. Architectural and Component documentation is one way for architects and developers to understand the systems, the components, and the environment. Because high turnover of staff is rather common and a lot of work is done by consultants, the supporting documentation is often the only source of information except the actual source code. Documents that should be present are; (1) system rationale which describes the objective of the entire system, (2) requirements specification which provides information on the exact requirements for the system as agreed between the user and the maintainer, (3) design document which provides descriptions of how the system requirements are implemented, of how the system is decomposed into a set of interacting program units, and the function of each program unit, (4) implementation document which provides descriptions of how detailed system design is expressed in a formal programming language, program actions in the form of intra-program comments, (5) system test plan which provides descriptions of how program units are tested individually and how the whole system is tested after integration, (6) acceptance test plan which describes the tests that the system must pass before users accept it, and (7) data dictionaries which contain descriptions of all terms that relate to the software system in action (Grubb and Takang 2003).

The System change environment and the Component change environment contain tools. The available tools have the intention of making all parts of the modification work easier.
There are many tools available to aid software development, i.e. supporting activities like code production, testing, and document generation. For the architect there are numerous modeling tools available for instance System Architect (Telelogic - IBM 2009), Troux 8 (Troux Technologies 2009), and Aris (IDS Scheer 2009). The system and component environment also includes infrastructure such as platforms. Platforms could for instance be; operating system platforms such as Linux and Windows XP or software platforms such as Java JDK or the .NET framework.

![Diagram of the modifiability metamodel presenting the main classes and reference slots.](image)

Fig. 2 The modifiability metamodel presenting the main classes and reference slots.

5.2 Attributes and their parents

In the previous subsection the classes and reference slots of the metamodel were presented, cf. Fig. 2. This subsection will focus on describing the attributes of these classes. This will be done by using so called metamodel views. The aim of these views is to present the metamodel in smaller segments that work together. Each view contains classes, reference slots, attributes, the causal dependencies, and the aggregating functions (introduced in section 3 and further explained in section 4.2). Together the views constitute the whole metamodel. The metamodel is however too large to fit into one single figure and still be readable, thus the focus on different views.

It was stated in the previous subsection that the main class is the change project class and as presented in the introduction the aim of the metamodel is to analyze modifiability. In the approach proposed in this paper modifiability is defined as change cost, thus the change project class contains the attribute Cost. The cost of a change project is measured as the number of man-hours needed to implement the modifications, $\{0, \ldots, \infty\}$.

Change projects are divided into architectural change activities and component change activities (cf. Fig. 3). Both types of activities have the attribute Cost, measured as number of
man-hours, \( \{0, \ldots, \infty\} \). The sum of the costs of these activities define the total change project cost. The second attribute of the activities is the Synchronization need attribute. The more systems, components, people involved, and the higher the coupling between them, the higher the need of synchronization among the different activities will be. Synchronization need is defined as the percentage of time spent on synchronization between different activities compared to each total activity cost, \( \{0, \ldots, 100\} \). (Boehm 1981; Grubb and Takang 2003; Bass et al. 1998)

The change activities are divided into technical changes which have two attributes, Change difficulty and Change size. Change difficulty is measured subjectively as \{Difficult, Normal, Easy\}. Change size for architecture changes is measured as the number of components involved in the change, \( \{0, \ldots, \infty\} \). Change size for components is measured as the number of lines of code involved in the modification, \( \{0, \ldots, \infty\} \). (International Organization for Standardization 2001; Chan et al. 1996; Grubb and Takang 2003; Pigoski 1997; April and Abran 2008)

The change organization (cf. Fig. 4) contains architects and developers. In order to estimate change cost an important attribute to measure in the change organization is the size, i.e. the Number of architects and Number of developers, \( \{0, \ldots, \infty\} \), involved in the modification work. (Oman et al. 1992; Chan et al. 1996; Grubb and Takang 2003; Pigoski 1997; Fenton and Pfleger 1997; Putnam and Myers 2003; April and Abran 2008)

The architects and developers both have the attributes Expertise and Time on project related to them. Expertise is measured in terms of Change project experience, Source code / design language experience, and System experience, where the three experience attributes are measured in number of years of experience, \( \{0, \ldots, \infty\} \). Time on project refers to the amount of time, in percentage, a person spend in the project compared to other parallel work, \( \{0, \ldots, 100\} \). (Oman et al. 1992; Chan et al. 1996; Boehm 1981; Grubb and Takang 2003; Pigoski 1997; Fenton and Pfleger 1997; April and Abran 2008; Smith 1999)

The change management process needs to be mature in order to provide the proper support for a project (cf. Fig. 5). Thus, the attribute important for this process is the Maturity attribute, which is measured by assessing Activities maturity, Number of assigned responsibilities, Number of documents, and Number of metrics (Simonsson 2008; The IT Governance Institute 2007; Oman et al. 1992; Boehm 1981; Grubb and Takang 2003; Pigoski 1997; April and Abran 2008; Kan 2003; Smith 1999). According to Cobit there are five activities in the change management process. The five activities are: (A) develop and implement a process to consistently record, assess and prioritize change requests, (B) assess impact and prioritize changes based on business needs, (C) assure that any emergency and critical change
follows the approved process, (D) authorize changes, and (E) manage and disseminate relevant information regarding changes. The activity maturity is measured using the maturity scale defined in Cobit. The scale has six steps; \( \{0 - \text{non existent}, 1 - \text{initial/ad hoc}, 2 - \text{repeatable but intuitive}, 3 - \text{defined process}, 4 - \text{managed and measurable}, \text{and} 5 - \text{optimized}\} \). Furthermore, each of the five activities needs to have an assigned responsibility, \( \{0, \ldots, 5\} \).

Cobit proposes 12 documents, \( \{0, \ldots, 12\} \), that should be available in the change management process, e.g. project management guidelines and detailed project plan, change process description, and change authorization. Cobit proposes 13 metrics, \( \{0, \ldots, 13\} \), that should be used during the modification work, e.g. reduced time and effort required to make changes, number of backlogged change requests, and number and type of emergency changes to the infrastructure components. (The IT Governance Institute 2007)

Documentation is crucial when it comes to understanding the systems, the components, and the environment involved in the change project. Therefore, the architecture documentation and the component documentation must be of high Quality. Documentation quality is defined by Availability, Completeness, Accuracy, Traceability, and Consistency. (Oman et al. 1992; Aggarwal et al. 2002; Grubb and Takang 2003; Pigoski 1997; April and Abran 2008; Smith 1999) A document can either be available or not available for the people involved in a project, thus the availability is measured digitally as \( \{\text{Available, Not available}\} \). Document completeness is the percentage of a document without missing information, \( \{0, \ldots, 100\} \). Accuracy refers to the percentage of a document being accurate, \( \{0, \ldots, 100\} \). Traceability is the percentage of a document with good traceability to the actual objects in the architecture or the components, \( \{0, \ldots, 100\} \). Document consistency is defined as the percentage of a document being uniform, standardized, and free from contradictions, \( \{0, \ldots, 100\} \).

**Fig. 4** The organizational view of the metamodel.

**Fig. 5** The documentational view of the metamodel.
Technical changes are implemented in either a system or a component and there are five attributes related to these, **Understandability**, **Internal coupling**, **Size**, **Complexity**, and **External coupling** (cf. Fig. 6). Understandability of a system or a component is measured as the percentage of time spent on trying to understand the system or component in question (in relation to the total time spent on each system/component), \( \{0, \ldots, 100\} \). Component size is measured in number of lines of code, and system size is defined as number of components, \( \{0, \ldots, \infty\} \). Complexity is measured subjectively as \{Complex, Medium, Not complex\}. The system external coupling attribute is defined as the number of actual relations between the systems divided by the number of possible relations between the systems, \( \{0, \ldots, 100\} \). System internal and component external coupling are measured as the number of actual relations between the components in the system divided by the number of possible relations between the components, \( \{0, \ldots, 100\} \). Component internal coupling is defined as the number of actual relations within the component divided with the number of possible relations, \( \{0, \ldots, 100\} \). (Oman et al. 1992; Matinlasi and Niemel 2003; Granja-Alvarez and Barranco-Garcia 1997; Chan et al. 1996; Aggarwal et al. 2002; Boehm 1981; Grubb and Takang 2003; Pigoski 1997; Bass et al. 1998; Fenton and Pfleger 1997; Putnam and Myers 2003; April and Abran 2008; Laird and Brennan 2006; Kan 2003; Smith 1999; Zuse 1997)

In order for the tools to support modification work and make it easier the tools have to be of high **Quality**, i.e. standardized, easy to use, and provide the right functionality. **Standardization level** is measured subjectively as \{Low, Medium, High\}, **Usability** is measured subjectively on a \{Low, Medium, High\} scale, and **Functional fit** is measured as the percentage of number of needed functions provided compared to total number of functions needed, \( \{0, \ldots, 100\} \). The system and component environment also includes infrastructure such as platforms. If the infrastructure **Quality** is poor, the modification work is likely to be impeded. Infrastructure quality is measured in terms of **Standardization level** subjectively defined as \{Low, Medium, High\} and **Availability** defined in percentage, \( \{0, \ldots, 100\} \). (Oman et al. 1992; Boehm 1981; Grubb and Takang 2003; Pigoski 1997; Fenton and Pfleger 1997; April and Abran 2008; Smith 1999)

Most attributes in the metamodel are measured objectively. There are however some attributes which have been chosen to be measured subjectively, for example system and component complexity. This is mainly a result of time restrictions but also due to the fact that in many cases the source code is not available for the modeler. In the best of worlds complexity would be measured objectively with for instance the Halstead complexity (Halstead 1975) or cyclomatic complexity (Grubb and Takang 2003), this is however often too expensive when you model and analyze many attributes. A discussion regarding costs and benefits of the different measures is thoroughly addressed in (Lagerström et al. 2009a).

To conclude, the metamodel classes, attributes and their relations are mainly based on academic literature. The conditional probability distributions related to the attributes are based on data collected in expert surveys and workshops. The estimation capabilities of the metamodel are tested in subsection 6.3 with data from 21 software change projects gathered in four multiple case studies conducted at large Nordic companies.

The metamodel depicted in Fig. 2 focuses on presenting the main classes and their reference slot names. Through out subsection 5.2 different views of the metamodel have been used to explain the aggregating classes and attributes. To summarize this section the metamodel in Fig. 7 presents the main classes and the causal dependency structure of their attributes. Thus, this figure neither includes reference slot naming and multiplicity nor does it include aggregating functions. However, it does contain additional information regarding the causal dependencies. These dependencies are numbered in order to relate them to the CPDs explained in section 4.2 and the data collection presented in section 6.2.
For the interested reader (Lagerström 2007) presents an early version of the modifiability metamodel.

6 Validation

The elements and structure of the modifiability metamodel as well as its estimation capabilities need to be evaluated and validated.

While the use of academic papers reflecting research serves as a good foundation for the metamodel creation, it is not completely trustworthy. There are several reasons for this. First, the scientific literature is not complete, so when creating a metamodel, it might be necessary
to fill in some blanks with hypotheses unsupported by the literature. Second, the scientific literature is not always coherent, so when creating a metamodel it might be unavoidable to make controversial choices. Third, the metamodeler might be biased and thus involuntarily introduce distortions. Expert validation of the metamodel attributes serves as a good function in minimizing these uncertainties.

Since the presented metamodel (PRM) is defined based on expert knowledge mapped to a three graded scale, there is a risk that the estimations are less accurate than wanted. Data based on 21 software change projects is used for testing the estimation capabilities of the metamodel.

The three most important questions are: 1) Are there attributes missing in the metamodel that should be added? 2) Are there superfluous attributes in the metamodel that could be removed? Together, these two questions determine whether the metamodel contains the appropriate elements, and they will be addressed in subsections 6.1 and 6.2 respectively. 3) Does the metamodel provide good estimation capabilities? The third question concerns the whole metamodel, both the qualitative and quantitative structure, i.e. the classes, reference slots, attributes, and causal dependencies, as well as the conditional probability distributions and aggregate functions. The estimation capability is validated by studying a number of change projects and by comparing the estimated cost with the actual cost outcome of the projects. This is addressed in subsection 6.3.

Finally, subsection 6.4 discusses the industrial feasibility of the modifiability metamodel.

6.1 Are there attributes missing in the metamodel?

The first question, “are there attributes missing in the metamodel?”, was posed to a number of experts, both academic experts in an online survey and industrial experts during workshops. Two workshops with industrial experts, with 6 and 27 experts respectively, were carried out. Of the 33 workshop experts 17 provided suggestions on attributes that could be missing in the metamodel. One online survey was sent out to academics in the field of software maintainability, this survey had 40 experts. Of the 40 survey experts 13 provided suggestions on missing attributes. Thus, the total number of experts were 73 and the total number of experts with suggestions on missing attributes were 30. The answers have been compiled into the list presented in Table 2.

Since there is no strong agreement among the 30 experts on which attributes are missing, the metamodel is considered to contain an appropriate set of attributes. The only attributes that two or more experts were missing are presented in Table 2. The attributes with the most votes in our workshops and survey had 3 persons out of the total 73 missing them, which is only 4.1%. If we consider the possibility that the 43 persons not missing any attributes at all are just convenient when answering the questionnaire, i.e. skips open-ended questions due to time restrictions, and instead restrict attention to the group that did at all suggest new attributes we have 3 votes out of a total 30 (10%). Still, this is a rather small amount of agreement on which attributes are missing in the metamodel. Thus, the answer to question one is: no, the proposed metamodel does not seem to be missing any attributes. Nevertheless, the attributes suggested are interesting and should be explored in future case studies and surveys.
Table 2 The workshop and survey results for the question "Which attributes are missing in the metamodel?".

<table>
<thead>
<tr>
<th>Class Attribute</th>
<th>No of answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management</td>
<td>3</td>
</tr>
<tr>
<td>Testing process</td>
<td>2</td>
</tr>
<tr>
<td>System</td>
<td>2</td>
</tr>
<tr>
<td>Requirements</td>
<td>3</td>
</tr>
<tr>
<td>Architecture</td>
<td>3</td>
</tr>
<tr>
<td>Change</td>
<td>2</td>
</tr>
<tr>
<td>Change</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class Attribute</th>
<th>No of answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management</td>
<td>3</td>
</tr>
<tr>
<td>Testing process</td>
<td>2</td>
</tr>
<tr>
<td>System</td>
<td>2</td>
</tr>
<tr>
<td>Requirements</td>
<td>3</td>
</tr>
<tr>
<td>Architecture</td>
<td>3</td>
</tr>
<tr>
<td>Change</td>
<td>2</td>
</tr>
</tbody>
</table>

6.2 Does the metamodel contain attributes that can be removed?

The second question, "are there superfluous attributes in the metamodel that could be removed?", was addressed by analyzing the strength of the causal dependencies between the metamodel attributes, as elicited from the experts. That is, if experts find that there is a strong causal connection between two attributes, than these attributes should be present in the metamodel. Conversely, if experts find that there is no causal connection between two attributes, than the parent attribute could be removed from the metamodel. Table 3 summarizes the causal dependencies found between the attributes of the metamodel.

There were in total 83 experts answering the questionnaire. 11 were industrial experts providing answers at two workshops and 72 were academic experts providing their answers via two online surveys.

The question posed to the experts both in the workshops and surveys was: “How large is the effect presented in the following statements?” Then the experts were provided with statements each corresponding to a causal dependency in the metamodel. See Fig. 7 for all corresponding causal relationships. The statements were all arranged as the following examples; “Change management process maturity affecting architectural change activities cost”, which corresponds to the relationship labeled as number 1 in the metamodel. “Number of architects affecting the architectural change activities cost”, corresponding to the causal dependency labeled as number 2. The answers provided by the experts were given on the following scale; High effect, Low effect, No effect, and I don’t know.

In the workshops and surveys the respondents were also provided with questions regarding their qualification as experts. In the workshops, two persons were excluded due to lack of expertise; both respondents stated that they had not enough experience in the field. Fourteen persons were excluded from the surveys. These either had too little experience (less than three years), they themselves said that they did not feel certain at all about their answers, or the answer I don’t know was given to more than 50 % of the questions asked.

Since the attributes in general have either high effect or high/low effect in relation to their parents, whereas very few had no effect, the attributes in the metamodel all seem useful for modifiability analysis. As can be seen in Table 3, no causal dependency has more than 17.1% of qualified respondent answers on No effect. We interpret these low percentages to indicate that there are no attributes in the metamodel with no effect on its causally related
attributes and by that the cost of making changes. Thus, the answer to question two is: no, the proposed metamodel does not seem to have any superfluous attributes that could be removed. However, some causal dependencies in the metamodel do have some answers on Low effect. These dependencies will be further validated in future change projects and surveys. Possibly one or two attributes can be removed or replaced, but this requires more research.

6.3 Does the metamodel provide good estimation capabilities?

The third question, "does the metamodel provide good estimation capabilities?", was addressed by studying four different multiple case studies. In the first case two projects within a Nordic consultancy firm were studied (projects A and B). The second case considered eight projects conducted within a large Nordic manufacturing company (projects C to J). Case three contained two projects at a large Nordic software and hardware vendor (projects K and L). In the fourth case nine change projects were studied at a large Nordic transportation company (projects M to U). In this paper one project, project M, from the case study at the Nordic transportation company is presented more thoroughly in order to illustrate the approach, cf. section 8.

Since all the studied projects are completed, data concerning their actual costs is obtainable. The estimated costs for the 21 projects are listed and compared to the actual costs of the projects, cf. Table 4. Along with this, the accuracy of each estimation is presented. The accuracy is calculated using the magnitude of the relative error (MRE), as defined by (Conte et al. 1985). Suppose \( E \) is the estimate of a value and \( A \) is the actual value, then the magnitude of the relative error for the estimate is

\[
MRE = \frac{|A - E|}{A}.
\]
Accuracy is then calculated as $1 - MRE$. This can be seen as the primary measure for the quality of the estimations the proposed metamodel is capable of. For instance, in our study of project M we can see that the actual cost turned out to be 5 810 man-hours. The estimated cost in turn, given the empirical data presented in Fig. 8, turned out to be 5 510 man-hours. Thus, the accuracy of the estimation can be calculated to be 95 %. Studying the other projects with respect to their actual and estimated costs we can see that 13 out of the 21 projects have an estimated cost within ranges of 75 % from the actual cost.

<table>
<thead>
<tr>
<th>Project size segment</th>
<th>Project</th>
<th>Actual cost (man-hours)</th>
<th>Estimated cost (man-hours)</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large</td>
<td>A</td>
<td>20 000</td>
<td>19 860</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>20 000</td>
<td>15 230</td>
<td>0.76</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>14 000</td>
<td>14 940</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>9 100</td>
<td>9 510</td>
<td>0.95</td>
</tr>
<tr>
<td>Medium</td>
<td>J</td>
<td>6 266</td>
<td>5 920</td>
<td>0.94</td>
</tr>
<tr>
<td></td>
<td>H</td>
<td>6 228</td>
<td>5 840</td>
<td>0.94</td>
</tr>
<tr>
<td></td>
<td>M</td>
<td>5 810</td>
<td>5 510</td>
<td>0.95</td>
</tr>
<tr>
<td></td>
<td>P</td>
<td>4 456</td>
<td>6 380</td>
<td>0.57</td>
</tr>
<tr>
<td></td>
<td>U</td>
<td>3 595</td>
<td>4 110</td>
<td>0.66</td>
</tr>
<tr>
<td>Small</td>
<td>L</td>
<td>3 200</td>
<td>3 085</td>
<td>0.96</td>
</tr>
<tr>
<td></td>
<td>K</td>
<td>3 200</td>
<td>2 195</td>
<td>0.69</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>3 000</td>
<td>2 480</td>
<td>0.83</td>
</tr>
<tr>
<td></td>
<td>I</td>
<td>2 440</td>
<td>2 170</td>
<td>0.69</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>2 300</td>
<td>2 240</td>
<td>0.97</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>2 054</td>
<td>1 915</td>
<td>0.90</td>
</tr>
<tr>
<td>Small (≤ 1 200 man-hours or less)</td>
<td>G</td>
<td>1 200</td>
<td>2 095</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>1 082</td>
<td>1 680</td>
<td>0.45</td>
</tr>
<tr>
<td></td>
<td>O</td>
<td>952</td>
<td>2 040</td>
<td>&lt;0</td>
</tr>
<tr>
<td></td>
<td>N</td>
<td>894</td>
<td>2 295</td>
<td>&lt;0</td>
</tr>
<tr>
<td></td>
<td>Q</td>
<td>454</td>
<td>2 005</td>
<td>&lt;0</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>262</td>
<td>1 805</td>
<td>&lt;0</td>
</tr>
</tbody>
</table>

Table 4 Studied projects with the actual costs, estimated costs, and accuracy of all conducted estimations.

One issue with the metamodel estimation concerns the project size: at what size does the metamodel no longer provide an acceptable estimation accuracy? The data from the 21 studied change projects provides an indication that the metamodel is to be employed in change projects over 2 000 man-hours. If the smallest projects, the ones under 2 000 man-hours are ruled out, then the accuracy measure spans from 57 % to 99 % with 13 out of 15 projects being estimated within a 75 % accuracy.

Thus the answer to question three is: yes, the proposed metamodel does seem to provide good estimation capabilities (at least for software change projects over 2 000 man-hours). However since the smallest projects (the ones under 1 200 man-hours in size) seem to be more difficult for the proposed metamodel to estimate, this will be addressed separately in future research.

For the interested reader a more elaborative validation is presented in (Lagerström et al. 2009b).
6.4 Industrial feasibility

The four case study companies were more than satisfied with the metamodel and the method it is a part of. At most companies the feasibility was evaluated internally at the companies in workshops and with presentations. One company has started an implementation of the approach in their change management process. Another company has asked for a project creating a tool for the modifiability metamodel. A third company is about to start a new case study testing the approach further.

7 Comparison with other models and methods

Since enterprise architecture is a discipline on the rise there are no alternatives within this modeling field that can be used for comparison. There are however other disciplines that have addressed the cost estimation problem. This section compares the estimation capabilities of three alternative models and methods with the modifiability metamodel.

According to (Conte et al. 1985) an acceptable accuracy level for an estimation method is something higher than or equal to 75 %. This notion is used to define a measure of prediction quality \( PRED \). In a set of \( n \) projects, let \( k \) be the number of projects where the accuracy is higher than or equal to \( q \). Then

\[
PRED(q) = \frac{k}{n}.
\]

According to (Conte et al. 1985) an estimation technique is acceptable if \( PRED(0.75) = 0.75 \). This means that in 75 % of the time the estimated values falls within 75 % of their actual values.

Three of the more well known and used models and methods are the COnstructive COst M0del (COCOMO) (Chulani et al. 1999; Boehm et al. 2000), function points (Matson et al. 1994) and planning poker (Moløkken-Østvold et al. 2008). As can be seen in Table 5 the modifiability metamodel seems to produce a prediction quality similar to COCOMO II.1997 and function points when considering all 21 projects. When considering the 15 projects of 2 000 man-hours and more the metamodel seems to have a better accuracy than all of these alternatives.

<table>
<thead>
<tr>
<th>Model / method</th>
<th>75 % accuracy, ( PRED(0.75) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>COCOMO II</td>
<td></td>
</tr>
<tr>
<td>1997</td>
<td>Before stratification 49%</td>
</tr>
<tr>
<td></td>
<td>After stratification 55%</td>
</tr>
<tr>
<td>2000</td>
<td>Before stratification 68%</td>
</tr>
<tr>
<td></td>
<td>After stratification 76%</td>
</tr>
<tr>
<td>Function points</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Model A 64%</td>
</tr>
<tr>
<td></td>
<td>Model B 68%</td>
</tr>
<tr>
<td>The modifiability metamodel</td>
<td></td>
</tr>
<tr>
<td></td>
<td>All projects 62%</td>
</tr>
<tr>
<td></td>
<td>All above 2 000 man-hours 87%</td>
</tr>
</tbody>
</table>

Table 5 Comparing the prediction quality of the modifiability metamodel, COCOMO II, and function points.

(Moløkken-Østvold et al. 2008) found that the mean estimation accuracy for planning poker was 82 %, cf. Table 6, while the mean estimation accuracy of the modifiability meta-
model is 88% (when studying the 15 project above 2 000 man-hours). As we can see they seem to provide rather similar estimation accuracies.

<table>
<thead>
<tr>
<th>Model / method</th>
<th>Mean accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>The modifiability metamodel, all above 2 000 man-hours</td>
<td>88%</td>
</tr>
<tr>
<td>Planning poker</td>
<td>82%</td>
</tr>
</tbody>
</table>

Table 6 Comparing the mean estimation accuracy of the modifiability metamodel and planning poker.

8 Models and analysis

The proposed metamodel is intended to be employed in decision situations regarding software change projects. Several case studies have been conducted based on the metamodel and the enterprise architecture analysis approach presented. In all, 21 software change projects at four different companies have been analyzed with the modifiability metamodel.

8.1 Case study information

One of the case studies was conducted at a large Nordic transportation company. Originally the multiple case study consisted of nine software change projects. In this paper, we detail one of them in order to describe how the created metamodel can be instantiated and used for cost analysis. For the interested reader (Lagerström et al. 2009b) presents a more detailed modeling and analysis section.

The company provides transportation services mostly within one of the Nordic countries, but also has some services in neighboring countries. They carry hundreds of millions of passengers every year and employ several thousand people.

The project studied, project M, was a project focusing on modifying the company’s public ticket webshop into an almost completely new version. The focus was on adding and changing functionality in the webshop software, making the purchase of tickets online easier, faster, more secure and more reliable. One especially important part of the project was to increase the usability with focus on the user interface for finding journeys, choosing ticket type, viewing price information, and printing pdf-tickets. The project included developing a new web flow and creating new integration solutions with the sales, journey planner, and ticket delivery systems. With these integrations it followed that there were numerous systems being modified within the project.

8.2 Data collection

The transportation company did not have an existing enterprise architecture function and no architecture models had been instantiated. Otherwise, information in existing models could have been used when instantiating project M. In the modifiability metamodel some classes and attributes are change project specific and always in need of new or at least updated information. However, the information regarding the involved systems, components, tools, infrastructure, documentation, personnel, and the change management process could
typically be found in already existing models. As mentioned in section 2.2 one issue to appre- reprehend is whether to trust the models, i.e. is the information presented in the models really updated. This usually depends on the enterprise architecture maintenance process (Fischer et al. 2007) and is not addressed further in this paper.

The data for project M was instead collected by interviewing and surveying people involved in the project, and by studying project documentation. Additional data was also collected with the development tools used at the company, for instance the size and component internal coupling measures. Based on the data collection an architecture model was instantiated for the project, illustrated in Fig. 8. The instantiated model presents project M on a high level, i.e. excluding the aggregating classes and their attributes.

8.3 Cost calculations

The probabilistic relational model based approach presented gives its values in the goal attribute as probabilities. For project M this means that the change cost has probability values
of being either High, Medium or Low. To enable estimations in man-hours, which is a more common and intuitive measure, the probabilities need to be transformed into actual man-hours. Transforming probabilities into cost, with a generally implementable cost interval, is not an easy task, and a significant part of our ongoing research. In this paper the problem of probability transformation is approached by the introduction of segments, cf. Table 7. These segments take into consideration whether the projects are seen as being of either Large, Medium or Small size. In order to reach the levels of accuracy presented in Table 4 the projects have to be fitted into one of the three segments before the estimation is done. This has so far only been done a posteriori and further research is being conducted to classify these more objectively and generally.

<table>
<thead>
<tr>
<th>Segment</th>
<th>High</th>
<th>Medium</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large</td>
<td>40 000</td>
<td>6 000</td>
<td>1 000</td>
</tr>
<tr>
<td>Medium</td>
<td>12 000</td>
<td>6 000</td>
<td>1 000</td>
</tr>
<tr>
<td>Small</td>
<td>5 000</td>
<td>2 500</td>
<td>1 000</td>
</tr>
</tbody>
</table>

Table 7 Cost intervals for categorization of change project size.
Once the project has been fitted into a segment, then it is possible to make a rather accurate cost estimation. Table 7 depicts the transformation from probabilities of High, Medium, and Low cost to number of man-hours for Large, Medium, and Small projects.

To be able to utilize the estimation capabilities of the metamodel for software change project cost analysis within the accuracy ranges in Table 4, a highly subjective prediction of the size of the project needs to be done. This high level prediction of the project size is preferably performed by project managers having experience of the project culture in the company where the metamodel is being applied. Here a simple categorization aims to determine if, dependent on the company, the largest projects commonly turn out to demand either 40 000, 12 000 or 5 000 man-hours. For example, if the largest projects in a company in general are considered to utilize man-hours closer to the range of 40 000 man-hours than 12 000 man-hours the project should be labeled as a Large project. This means that the probabilities of a project being High, Medium, or Low in the metamodel are mapped to the man-hour levels 40 000, 6 000 and 1 000 respectively. If the project instead had been determined to be of Medium size the probabilities would be mapped to the levels 12 000, 6 000 and 1 000.

In a future version of the presented metamodel there will hopefully not be a need for the project managers to match projects with the segments presented in Table 7. General cost levels suitable for change cost estimations independent of the project manager’s subjective size segmentation of the projects will be obtained through empirical studies where data for calibrating the change cost attribute will be used.

8.4 Case study conclusions

Based on project M the following points can be emphasized: 1) the EA metamodel did provide an estimation within 95 % accuracy. 2) The metamodel enables early finding and highlighting of risks. For instance, the instantiated model for project M shows that the change difficulty is high for both the architectural and component changes. It also shows that the involved systems are complex, that there are as many as 1188 components involved in the change, and that the systems involved are tightly coupled together.

A decision maker, typically a project leader within software change projects, will be provided with three sorts of valuable information when utilizing the presented metamodel. Firstly, the expected costs of a set of change projects can be estimated. This enables a more rational decision making concerning what parts of a company’s project portfolio should be prioritized. Furthermore, it is possible for a decision maker to test different scenarios regarding the objects and attributes in the instantiated architectural model in order to try to lower the cost of a specific project. This could for example be realized by involving other developers having more suitable experience within the chosen design specific language in the project.

Secondly, when a project has been chosen from the portfolio to be initiated, the architectural model will be able to aid the project manager in the planning phase of the project. The planner can for instance get assistance to choose both architectural and development team size, as well as elaborate how the team expertise will affect the outcome.

Thirdly, the instantiated architectural model can be used for conducting risk analysis. The model is for example able to reveal which parts of the project carry a high risk of cost occurrence. Hence, action plans can be set up accordingly to mitigate the occurrence of these risks. This enables the resources chosen for the project to be optimized for the
project specific change activities. Hence, the risk of the project exceeding its given budget and timeframes are somewhat more controllable.

9 Discussion

During the research work, several issues concerning the validity have been addressed. Multiple sources in both the development phase and the validation phase were used. Key stakeholders have been addressed for reviews of drafts. A chain of evidence was established. Theory was developed and used. There are however some threats to the validity. These threats mainly concern the 21 change projects studied in the validation phase when focusing on the estimation capabilities. Since there was no possibility of studying projects from start to end most data gathered in these case studies was collected after the projects were finished. Thus providing a final cost of the projects for comparison, but also inferring an uncertainty of the a priori value of the data. Another issue being a threat to the validity concerns the scales used when collecting and analyzing data, especially the transformation between these scales. The cost interval segmentation used when estimating the costs is also a validity issue since this so far only has been done after the projects were finished.

Reliability was achieved by documenting the research work during all phases. All case studies have their own reports describing data collection and analysis. The aim has been to operationalize as many steps as possible, e.g. conducting surveys with predefined choices and using methods with well defined guidelines and rules, thereby allowing other investigators to repeat the work.

9.1 Future work and implications

To the industrial practitioner, the present paper assists the enterprise modeling effort when the concern is focused on modifiability and enterprise software change. If the metamodel is employed in the beginning of change projects the modeler will get cost estimations and the possibility to highlight risks early.

Since large contemporary enterprises all have different concerns, systems, organizations etc, the presented metamodel might need to be tailored to fit each company (Brinkkemper 1996; 2000). The idea for the industrial practitioner would thus be to employ the modifiability metamodel as a base together with the method presented in (Lagerström et al. 2009d). This would provide the user with a stakeholder concerns tailored metamodel.

To the EA tool industry, the present paper hints to potential new features or products. A tool incorporating the provided metamodel would give qualitative and quantitative support to the user’s modeling effort. The current market for EA tools does not explicitly consider different analyses, especially not with focus on modifiability and change cost.

To the scientific community, the presented metamodel combines the approach of enterprise architecture modeling for analysis with the approaches of modifiability analysis and change cost estimation. These communities can benefit from this work as well as continue contributing in the combined area by extending/improving the metamodel and the methods utilizing it. As discussed in the validation section there are some metamodel elements that after further research perhaps could be reconsidered, i.e. removed or changed. Also, in the surveys and case studies some new elements have been suggested to be included in the metamodel. Before any such additions can be made some additional research is needed. Since
the current version of the metamodel mainly focuses on change organization, change environment, documentation and software system related issues, special focus could be on the business related elements of enterprise architecture. Many business issues are now implicitly included in the change difficulty attribute of the technical changes class. New classes to add and test might be; management, business organization, requirement specification and business process.

Another part of the metamodel that could be improved is the final step of the cost calculation. In the modifiability metamodel presented in this paper, the cost translation from probabilities to man-hours is made on a change project level. We do however believe that the cost transformation can be improved if the focus shifts from project to activity, thus translating the cost for each change activity instead of the whole project. This will however need more data before it can be implemented.

Since the size segmentation used when calculating the estimated cost has been calibrated based on data collected from already finished projects this might be biased. A next step is thus to follow projects from start to end in order to collect more accurate data and to improve the calibration of the segmentation.

In this paper expert validation and case studies have been used for validating the modifiability metamodel. Beside these, there are other methods that in the future could be used to further test and validate the metamodel. An appropriate method could be structural equa-
tional modeling (SEM) (Warner 2008). SEM is mainly used for testing causal relationships and it requires a larger data set than the one available in this study.

10 Conclusions

Enterprise architecture models can be used in order to increase the general understanding of enterprise systems and specifically to perform various kinds of analysis. This paper proposes a metamodel for enterprise systems modifiability analysis, i.e. assessing the cost of making changes to enterprise-wide systems. The enterprise architecture metamodel is formalized using probabilistic relational models, enabling the combination of regular entity-relationship modeling aspects with means to perform enterprise architecture analysis. The presented metamodel contains classes such as change activities, architects and developers, systems and their components, documentation, infrastructure, and change management process. Each class has a set of attributes related to it. For instance, the class System has the attributes understandability, internal and external coupling, complexity, and size. These attributes are causally related to each other, providing the user with analysis capabilities under conditions of uncertainty.

The paper discusses the validity of the modifiability metamodel based on data collected in workshops and surveys with both academia and industry. The studied data indicates that the metamodel contains the appropriate elements. Data was also collected in 21 software change projects by employing the metamodel. This data was used in order to validate the metamodel's estimation capability, i.e. how well the metamodel estimates software change cost. The metamodel produced estimates within a 75 % accuracy for 13 out of 15 of the projects above 2 000 man-hours. However, the metamodel seems to provide less accurate estimations for the smallest projects, i.e. the ones with a cost of 1 200 man-hours and less.

Furthermore, the modifiability metamodel is instantiated based on a case study at a large Nordic transportation company, showing the applicability of the proposed metamodel.
References


Chapter 6

Paper E: Architecture Analysis of Enterprise Systems Modifiability – Models, Analysis, and Validation
Abstract

Enterprise architecture (EA) models can be used in order to increase the general understanding of enterprise systems and to perform various kinds of analysis. This paper presents instantiated architectural models based on a metamodel for enterprise systems modifiability analysis, i.e. for assessing the cost of making changes to enterprise-wide systems. The instantiated architectural models detailed are based on 21 software change projects conducted at four large Nordic companies. Probabilistic relational models (PRMs) are used for formalizing the EA analysis approach. PRMs enable the combination of regular entity-relationship modeling aspects with means to perform enterprise architecture analysis under uncertainty. The modifiability metamodel employed in the analysis is validated with survey and workshop data (in total 110 experts were surveyed) and with the data collected in the 21 software change projects. Validation indicates that the modifiability metamodel contains the appropriate set of elements. It also indicates that the metamodel produces estimates within a 75% accuracy in 87% of the time and has a mean accuracy of 88% (when considering projects of 2,000 man-hours or more).

Key words: Enterprise architecture, Software change cost estimation, Software modifiability, Metamodel, Probabilistic relational models

1. Introduction

Managing software systems today is a complex business. In order to achieve effective and efficient management of the software system landscape it is essential to be able to assess the current status of system qualities such as availability, performance, security, and modifiability, as well as estimate their status in different future scenarios. Estimation of these qualities is however a great challenge that to
a large extent can be addressed by introducing models as a means of abstraction. This is achieved with enterprise architecture modeling.

The purpose of this paper is to: 1) present instantiated architectural models based on the metamodel supporting analysis of systems modifiability, i.e. estimation of enterprise software system change cost. The created models are instantiations of 21 software change projects studied in multiple case studies conducted at four large Nordic companies, 2) discuss the validity of the modifiability metamodel elements by studying data collected in workshops and surveys (in total 110 experts were surveyed) and 3) discuss the estimation capabilities of the metamodel by studying the actual and estimated costs of the 21 software change projects.

The contribution of this paper is the instantiation and validation of the modifiability metamodel. (Lagerström et al. 2009b) thoroughly describes this metamodel and (Lagerström et al. 2009a) details the creation method which it is based on. Thus, the research work conducted is divided into three papers with different focus and contributions.

1.1. Enterprise architecture

In recent years, Enterprise Architecture (EA) has become an established discipline for business and software system management (Ross et al. 2006). EA describes the fundamental artifacts of business and IT as well as their interrelationships (Zachman 1987; Lankhorst 2005; Ross et al. 2006; Winter and Fischer 2007; The Open Group 2009). Architecture models constitute the core of the approach and serve the purpose of making the complexities of the real world understandable and manageable (Winter and Fischer 2007). EA ideally aids the stakeholders of the enterprise to effectively plan, design, document, and communicate IT and business related issues, i.e. they provide decision support for the stakeholders (Kurpjuweit and Winter 2007).

In relation to supporting decisions, a key underlying assumption of EA models is that they should provide some more aggregated knowledge than what was merely put into the model in the first place. Software application architecture, for instance, does not only keep track of the set of systems in an enterprise and their internal relationships, it also provides information about the dependencies between the systems. More broadly, the dependencies between the business and the software systems are covered in an EA. Conclusions can therefore for instance be drawn about the consequences in the enterprise given that one specific system is unavailable.

Enabling this type of analysis is extremely important for providing value of EA to its stakeholders. Unfortunately, EA frameworks rarely explicitly state neither what kinds of analysis that can be performed given a certain model nor the details on how the analysis should be performed (Johnson et al. 2007; Franke et al. 2009). Another permeating problem in EA modeling is the uncertainty that is related to
the development of the model. For instance, if a model is the result of a very thorough and recent investigation or a quick read-through of old documents, the resulting quality of the model will differ. In turn this will impact the quality of the decision support the model offers for its various stakeholders. Are all the software systems in the model still in use, is the data flow still as depicted, and is the process structure illustrating what is really happening? This kind of uncertainty is not addressed by today’s EA frameworks. Again the user of the models is simply left to her subjective best knowledge or gut feeling when estimating to what extent the EA model, and the analyses based on it, can and should be trusted.

This paper employs a formalized approach to enable analysis of EA models. The approach also copes with empirical uncertainties by not considering the information in EA models as fixed but rather as probabilities. The underlying fundamental formalism in the approach is called Probabilistic Relational Models (PRMs) (Friedman et al. 1999), which in turn employs statistical mathematics of Bayesian networks (Jensen 2001; Neapolitan 2003).

1.2. Enterprise system modifiability

As discussed in the previous subsection, enterprise architecture models can be used to analyze different system qualities and provide information for the decision maker regarding different scenarios. In this paper the focus will be on enterprise software system modifiability, i.e. the cost of making changes to enterprise-wide software systems. Cost is here defined as man-hours spent on a change project.

Business environments today need to progress and change rapidly to keep up with evolving markets. Most business processes are supported by software systems and as the business processes change, the systems has to be modified in order to continue supporting the processes. Modifications include extending, deleting, adapting, and restructuring the enterprise software systems (Bass et al. 1998). Modification efforts can range from adding a functional requirement in a single system to implementing a service oriented architecture for a complete enterprise.

An essential issue with today’s software systems is that many of them are interconnected, thus a modification to one system may cause a ripple effect among other systems. It is also common that the systems have been developed and modified during many years. Making further changes to these systems might require a lot of effort from the organization, for example due to a large amount of previous modifications implemented ad hoc. Problems like these raise questions for IT decision makers such as: Is there enough documentation describing the system? Has the documentation been updated correctly after each modification? Is the source code easy to understand? Or, which systems are interconnected?

According to (IEEE Standards Board 1990), maintenance and maintainability can be interchangeably used with the terms modification and modifiability. Maintenance is in (IEEE Standards Board 1990) defined as the ease with which
a software system can be modified, maintainability is defined as the process of modifying a software system or component. With this confirmed we can lean towards and enlight several studies indicating that modification work is the phase in a software system’s lifecycle that consumes the greatest portion of resources: (Harrison and Cook 1990) report that over 70% of the software budget is spent on maintenance, (Pigoski 1997) refers to studies stating that the maintenance cost, relative to the total life cycle cost of a software system, has been increasing from 40% in the early 1970s up to 90% in the early 1990s, and (Jarzabek 2007) states that “the cost of maintenance, rather than dropping, is on the increase”.

The activities of modifying enterprise systems are typically executed in projects, and IT decision makers often find it difficult to estimate and plan their change projects. Thus, a large proportion of the projects aiming to modify a software system environment fail, i.e. the projects tend to take longer time and cost more than expected. (Laird and Brennan 2006) declare that 23% of all software projects are cancelled before completion, whereas of those completed only 28% were delivered on time, and the average software project exceeded its budget by 45%. This can often occur due to lack of information about the systems being changed. According to (Laird and Brennan 2006), software engineers must be able to understand their activities, as well as manage the risks, through estimation and measurement. Therefore, it would be useful for IT decision makers to gather more information in a structured manner and use this information to analyze how much effort a certain modification of an enterprise software system would require.

This paper will address these issues of software change by employing enterprise architecture modeling. Instantiated architectural models for estimation of software change project cost, i.e. for systems modifiability analysis, will be presented.

1.3. Enterprise architecture for modifiability analysis

As stated in subsection 1.1, the exact procedure or algorithm for how to perform a certain analysis given an architecture model is very seldom provided by EA frameworks. Most frameworks do however recognize the need of providing special purpose models and provide different viewpoints intended for different stakeholders. Unfortunately, most viewpoints are designed from a model entity point of view, rather than a stakeholder concern point of view. Thus, assessing a quality such as the modifiability of a system is not something that is performed in a straightforward manner. The Department of Defense Architecture Framework (DoDAF) for instance provides products (i.e. viewpoints) such as systems communications description, systems data exchange matrix, and operational activity model (Department of Defense Architecture Framework Working Group 2007). These are all viewpoints based on a delimitation of elements of a complete metamodel, and they are not explicitly connected to a certain stakeholder or purpose.
The Zachman framework connects model types describing different aspects (Data, Function, Network, People, Time, and Motivation) with very abstractly described stakeholders (Strategists, Executive leaders, Architects, Engineers, and Technicians) (Zachman 1987; 2009), but does not provide any deeper insight how different models should be used. The Open Group Architecture Framework (TO-GAF) is explicitly stating stakeholders and concerns for each viewpoint they are suggesting (The Open Group 2009). However, neither the exact metamodel nor the mechanism for analyzing the stated concerns, are described.

In relation to modifiability, the most appropriate viewpoints provided by TO-GAF would arguably be the Software Engineering View, the Systems Engineering View, the Communications Engineering View, and the Enterprise Manageability View. In the descriptions of these views one can find statements such as; "the use of standard and self-describing languages, e.g. XML, are good in order to achieve easy to maintain interface descriptions". However, the exact interpretation of such statements when it comes to architectural models or how it relates to the modifiability of a system as a whole, is left out. Moreover, these kinds of "micro theories" are only exemplary and do not claim to provide a composed theory for modifiability or similar concerns.

Other, more formalized analysis mechanisms for enterprise architecture models, may be found in e.g. (Lankhorst 2005) for performance and availability, for software architecture languages in (Allen 1997) where for instance dead-lock and interoperability analyses are provided, Architecture Analysis and Design Language (AADL) (Society of Automotive Engineers 2009) provides availability, security, and timeliness analyses, and UMLsec (Jürjens 2005) provides security analysis. None, however, offer architecture models for software system modifiability analysis.

Since there are no EA-frameworks or metamodels focusing on modifiability analysis available the present paper aims at filling this gap in enterprise architecture.

1.4. Outline

The remainder of the paper is structured as follows: Section 2 presents the probabilistic relational models which serve as the underlying formalism for the enterprise architecture models presented. The subsequent section describes the method employed for designing the modifiability metamodel, i.e. metamodel for enterprise software system change cost estimation. Next, in section 4 the modifiability metamodel is presented. The qualitative part of the metamodel is based on existing academic literature and the quantitative part of information gathered in expert surveys and workshops. Section 5 contains information regarding the modeling approach for instantiating the architectural models. A set of instantiated architectural models for software change project cost based on four multiple
case studies are described in section 6 in order to illustrate the applicability of the EA analysis approach. In section 7 the metamodel for modifiability is validated by considering the correctness of the qualitative structure and the estimation capabilities of the quantitative structure. The qualitative structure is validated with survey and workshop data from 110 experts. Actual costs and estimated costs from 21 software change projects are compared in order to validate the quantitative structure. COCOMO, function points, and planning poker are other available estimation methods. Section 8 compares the estimation capabilities of the modifiability metamodel with the capabilities of these methods. Discussion and research outlook is presented in section 9. Finally, section 10 summarizes the paper with conclusions.

2. Probabilistic relational models

As stated in the introduction Probabilistic Relational Models (PRMs) serve as the underlying formalism for the enterprise architecture models described in this paper. Previously proposed formalisms such as the Extended influence diagrams are presented in (Johnson et al. 2007; Lagerström 2007).

A PRM (Friedman et al. 1999) specifies a template for a probability distribution over an architecture model. The template describes the metamodel for the architecture model, and the probabilistic dependencies between attributes of the architecture objects. A metamodel acts as a pattern for the instantiation of models. In other words, a metamodel is a description language used when creating models (Lankhorst 2005; Johnson and Ekstedt 2007; Kurpjuweit and Winter 2007; The Open Group 2009). A PRM, together with an instantiated architecture model of specific objects and relations, defines a probability distribution over the attributes of the objects. The probability distribution can be used to infer the values of unknown attributes, given evidence of the values of a set of known attributes.

An architecture metamodel \( \mathcal{M} \) describes a set of classes, \( \mathcal{X} = X_1, \ldots, X_n \). Each class is associated with a set of descriptive attributes and a set of reference slots. The set of descriptive attributes of a class \( X \) is denoted \( \mathcal{A}(X) \). Attribute \( A \) of class \( X \) is denoted \( X.A \) and its domain of values is denoted \( V(X.A) \). For example, a class DeveloperTeam might have the descriptive attribute Expertise, with domain \{High, Medium, Low\}. The set of reference slots of a class \( X \) is denoted \( \mathcal{R}(X) \). We use \( X.\rho \) to denote the reference slot \( \rho \) of class \( X \). Each reference slot \( \rho \) is typed with the domain type \( \text{Dom}[\rho] = X \) and the range type \( \text{Range}[\rho] = Y \), where \( X;Y \in \mathcal{X} \). A slot \( \rho \) denotes a relation from \( X \) to \( Y \) in a similar way as Entity-Relationship diagrams. For example, we might have a class DeveloperTeam with the reference slot IsAResourceOf whose range is the class ChangeOrganization.

An architecture instantiation \( \mathcal{I} \) (or an architecture model) specifies the set of objects of each class, the values for the attributes, and the references of the objects.
For example, Fig. 6 presents an instantiation of the change project metamodel of Fig. 4. It specifies a particular set of changes, systems, documents, etc., along with their attribute values and references. For future use, we also define a relational skeleton $\sigma_r$ as a partial instantiation which specifies the set of objects in all classes as well as all the reference slot values, but not the attribute values.

A probabilistic relational model $\Pi$ specifies a probability distribution over all instantiations $\mathcal{I}$ of the metamodel $\mathcal{M}$. This probability distribution is specified similar to a Bayesian network (Jensen 2001), which consists of a qualitative dependency structure and associated quantitative parameters.

The qualitative dependency structure is defined by associating with each attribute $X.A$ a set of parents $Pa(X.A)$. Each parent of $X.A$ is defined as $X.\tau.B$ where $B \in \mathcal{A}(\tau)$ and $\tau$ is either empty, a single slot $\rho$ or a sequence of slots $\rho_1, \ldots, \rho_k$ such that for all $i$, $\text{Range}[\rho_i] = \text{Dom}[\rho_{i+1}]$. For example, the attribute Cost of class ChangeProject may have ArchChangeActivity.System.Understandability as parent, thus indicating that the cost of a prospective software modification project depends on the understandability of the systems changed in the architecture. Note that $X.\tau.B$ may reference a set of attributes rather than a single one. In these cases, we let $A$ depend probabilistically on some aggregate property over those attributes, such as the logical operations AND, OR, and NOR. In this paper we use the arithmetic operations SUM and MEAN as aggregate functions. For instance, if there are several systems changed in a modification project, we might aggregate each systems’ understandability into a mean understandability of the whole architecture.

Considering the quantitative part of PRMs, given a set of parents for an attribute, we can define a local probability model by associating a conditional probability distribution (CPD) with the attribute, $P(X.A|Pa(X.A))$.

We can now define a PRM $\Pi$ for a metamodel $\mathcal{M}$ as follows. For each class $X \in \mathcal{X}$ and each descriptive attribute $A \in \mathcal{A}(X)$, we have a set of parents $Pa(X.A)$, and a CPD that represents $P_{\Pi}(X.A|Pa(X.A))$.

Given a relational skeleton $\sigma_r$ (i.e. a metamodel instantiated to all but the attribute values), a PRM $\Pi$ specifies a probability distribution over a set of instantiations $\mathcal{I}$ consistent with $\sigma_r$:

$$P(\mathcal{I}|\sigma_r, \Pi) = \prod_{x \in \sigma_r(X)} \prod_{A \in \mathcal{A}(x)} P(x.A|Pa(x.A))$$

where $\sigma_r(X)$ are the objects of each class as specified by the relational skeleton $\sigma_r$.

A PRM thus constitutes the formal machinery for calculating the probabilities of various architecture instantiations. This allows us to infer the probability that a certain attribute assumes a specific value, given some (possibly incomplete) evidence of the rest of the architecture instantiation. In addition to express and infer
3. Creating the modifiability metamodel

Creating a good enterprise architecture metamodel is not trivial. Obviously, it is important that the metamodel is tailored for the management tasks it should support, i.e. what kind of analysis the metamodel will be subjected to. For instance, if one seeks to employ an enterprise architecture model for evaluating IT/business alignment, the information required from the model differs radically from the case when the model is used to evaluate the modifiability of an enterprise software system. This section aims at presenting the method employed when the modifiability metamodel was designed. The method focuses on creating a special kind of metamodel called probabilistic relational model (PRM). As described in section 2, PRMs contain attributes which are causally related. These are used for analysis and estimation with probabilities, while regular entity-relationship models and UML-class diagrams do not. For the interested reader a more detailed description of the method is presented in (Lagerström et al. 2009a).

3.1. Method for creating the qualitative part of the metamodel

This subsection presents the method for creating the qualitative part of the modifiability metamodel, i.e. the classes, reference slots, attributes and their parents.

The method employed for creating metamodels for decision support is of an iterative nature and focuses on finding a set of appropriate \textit{a priori} measures for a chosen goal, i.e. finding measures with high correlation with and causal influence on the selected goal. This can be done in several ways, for instance by studying research literature, conducting experiments, doing case studies, or using expert opinions.

The first step in the method is to select the goal that the metamodel under design is supposed to support. In this paper the goal considered is modifiability, i.e. change cost. Then, variables affecting the goal, and variables affecting previously identified variables are identified and causally related to each other. This iterative process continues until all paths of variables, and causal relations between them, have been broken down into variables that are directly controllable by the decision maker, see part 1 and 2 of Fig. 1. The process of finding variables affecting modifiability is supported by knowledge elicitation guidelines and control steps. These have been described in (Johnson et al. 2007; Lagerström et al. 2007; 2009a). The result of the iterative process is a goal break-down. The goal break down is based on scientific knowledge, exhibiting variables that are controllable by the decision maker.
maker and causally linked to a well-defined goal, e.g. as in part 3 of Fig. 1. The next step is to translate these into metamodel classes, attributes, and reference slots. The attributes of the metamodel classes correspond to the variables found in the goal break-down part of the method. The goal break-down example presented in part 3 of Fig. 1 corresponds to the metamodel example presented in part 4 of the same figure.

Figure 1: The iterative goal break-down process, a resulting goal break-down example, and the corresponding metamodel.

By using the guidelines for metamodel design, the qualitative part of a modifiability analysis metamodel was finally obtained. This qualitative part is based on existing academic literature. Sources were chosen based on how well they fit the research topic (modifiability, maintainability or change cost analysis) and that they are written in an academic way (either peer-reviewed papers or well–cited books). Fig. 2 and Fig. 3 present two views of the modifiability metamodel. A view contains classes, reference slots, attributes, causal dependencies, and aggregating functions. The metamodel view illustrated in Fig. 4 contains the main classes and attributes, as well as the causal structure of the metamodel. However, this view neither presents the aggregation functions nor the multiplicities.

3.2. Method for creating the quantitative part of the metamodel

This subsection presents the second part of the metamodel creation method, defining the quantitative part of the metamodel. That is, to define the conditional probability distributions (CPDs) related to each attribute.

The metamodel creation method employs a knowledge elicitation approach previously published in (Lagerström et al. 2009a;c). This approach takes expert opinions into consideration when defining the conditional probability distributions without the need of introducing the experts to the concepts of conditional probabilities and PRMs. The algorithms used for defining the CPDs in the metamodel are based on the effect one attribute \( x \) has on a related attribute \( y \). In the modifiability case the effect was found by using a questionnaire among experts, during
both workshops and in an online survey. The effect is measured on an ordinal scale with three states High effect, Low effect, and No effect.

The resulting CPDs will look like the following example between five attributes and their joint effect on Component change activities cost, cf. Table 1.

<table>
<thead>
<tr>
<th>Change mgmt. process maturity (17)</th>
<th>Not mature</th>
<th>...</th>
<th>Very m.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change org. no of developers (18)</td>
<td>Many</td>
<td>...</td>
<td>Few</td>
</tr>
<tr>
<td>Developer team expertise (19)</td>
<td>Low</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Tech. changes to comp. change diff. (21)</td>
<td>Difficult</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Comp. change act. synch. need (20)</td>
<td>High</td>
<td>Med.</td>
<td>Low</td>
</tr>
<tr>
<td>Component change activities cost</td>
<td>High</td>
<td>0.88</td>
<td>0.73</td>
</tr>
<tr>
<td></td>
<td>Medium</td>
<td>0.06</td>
<td>0.21</td>
</tr>
<tr>
<td></td>
<td>Low</td>
<td>0.06</td>
<td>0.06</td>
</tr>
</tbody>
</table>

Table 1: The resulting conditional probability distribution for the attribute Cost of the class Component change activities affected by the Change management process maturity, Number of developers in the change organization, Developer team expertise, Change difficulty of the technical changes to components and Synchronization need of the component change activities.

As described in section 2 there are some attributes that are related as aggregates rather than by causality. In these cases, the probabilistic dependency is on some aggregate property over those attributes, such as the arithmetic operations SUM and MEAN. In the metamodel, cf. Fig. 3 – Fig. 4, there are several classes having underlying classes that serve as Is-a-part-of classes. E.g. in the system view, cf. Fig. 3, the system change environment consists of a number of tools. Here, the quality of each tool aggregates by the MEAN operation to the quality of the change environment. Since the aggregate functions are not based on causality but rather on aggregates of attributes, these CPDs are not defined based on data and can not be found with experiments. These aggregate functions are definitions decided by the metamodeler and are in the modifiability case intuitively set as SUM or MEAN.

4. The modifiability metamodel

This section presents an enterprise architecture metamodel for modifiability analysis. Outlined are the classes of the metamodel, its reference slots and attributes. The interested reader is referred to (Lagerström et al. 2009b) for a complete description of the metamodel. The main elements of the metamodel textually described in this section are depicted in Fig. 4.

The suggested metamodel for modifiability analysis focuses on software systems and their surrounding environment involved in or affected by modifications
implemented in a change project. The main element of the metamodel is the Change project class. Since modifiability in this paper is defined as cost of change the natural attribute of the change project class is Cost.

Change projects are divided into Architectural change activities and Component change activities (Boehm 1981; Bass et al. 1998; Grubb and Takang 2003). Architectural change activities represent modifications on an architectural level, while a component change activity concerns modifications of a single component of an application. The two activity types both have Cost as their main attribute. The sum of these costs constitute the total cost of a change project. Moreover, these two change activities are attributed with Synchronization need as a measure of the need for alignment between activities involved in the change project.

The two sorts of change activities are supported by a Change management process possessing the attribute Maturity, emphasizing the importance of having a rigid and proper support function for the change project being conducted (Boehm 1981; Oman et al. 1992; Pigoski 1997; Smith 1999; Grubb and Takang 2003; Kan 2003; The IT Governance Institute 2007; April and Abran 2008; Simonsson 2008).

Component change activities and architectural change activities perform technical changes to software components and system architecture respectively. These changes are illustrated in the metamodel by the classes Technical changes to components and Technical changes to architecture. The two classes are attributed with Change difficulty and Change size (Chan et al. 1996; Pigoski 1997; International Organization for Standardization 2001; Grubb and Takang 2003; April and Abran 2008).

Fig. 2 presents a view illustrating the change project class, the classes for change activities and technical changes with their respective attributes, reference slots, causal structures, and aggregating functions.

![Figure 2: The project and activity view of the modifiability PRM.](image)

Change activities are executed either by architects, belonging to an Architect team, or by developers belonging to a Developer team. Attributes needed for
cost estimation related to these classes are *Time on project* and *Expertise* (Boehm 1981; Oman et al. 1992; Chan et al. 1996; Fenton and Pfleger 1997; Pigoski 1997; Smith 1999; Grubb and Takang 2003; April and Abran 2008).

The documentation supporting the change activities is written according to the change management process. The set of documentation is modeled as either *Architectural documentation* or *Component documentation* depending on its content. The most important aspect of the documentation is its quality, hence *Quality* is the main attribute for both types of documentation (Oman et al. 1992; Pigoski 1997; Smith 1999; Aggarwal et al. 2002; Grubb and Takang 2003; April and Abran 2008).

The architect team and developer team are modeled as resources of the *Change organization* in the metamodel. The relevant attributes related to cost estimation for change organizations are the total number of architects and developers involved in the project. Thus *No of architects* and *No of developers* serve as the two attributes of the change organization (Oman et al. 1992; Chan et al. 1996; Fenton and Pfleger 1997; Pigoski 1997; Grubb and Takang 2003; Putnam and Myers 2003; April and Abran 2008).

The classes technical changes to architecture and components each work in their respective environment, defined in the metamodel by the classes *System change environment* and *Component change environment*. These two classes both have the attributes *Quality of tools* and *Quality of infrastructure* (Boehm 1981; Oman et al. 1992; Fenton and Pfleger 1997; Pigoski 1997; Smith 1999; Grubb and Takang 2003; April and Abran 2008).

The modifications in a change project are either implemented in *Systems* or *Components*. These classes are both attributed with *Understandability*, *Internal coupling*, *Size*, *Complexity*, and *External coupling* (Boehm 1981; Oman et al. 1992; Chan et al. 1996; Fenton and Pfleger 1997; Granja-Alvarez and Barranco-Garcia 1997; Pigoski 1997; Zuse 1997; Bass et al. 1998; Smith 1999; Aggarwal et al. 2002; Grubb and Takang 2003; Kan 2003; Matinlasisi and Niemel 2003; Putnam and Myers 2003; Laird and Brennan 2006; April and Abran 2008). In Fig. 3 a view illustrating systems, components, and the change environment classes is presented.

The metamodel depicted in Fig. 4 presents the main classes, their reference slots, and the causal dependency structure of their attributes. Thus, this figure neither includes reference slot multiplicity nor does it include aggregating functions. However, it does contain additional information regarding the causal dependencies. These dependencies are numbered in order to relate them to the CPDs explained in section 3.2 and the data collection presented in section 7.2. Two views of the metamodel, cf. Fig. 2 and Fig. 3, have been used to illustrate the aggregating classes and attributes. There are other views of the metamodel, these are presented in (Lagerström et al. 2009b). Together the views constitute the whole
metamodel. It is however to large to fit into one single figure and still be readable, thus the focus on different views.

Figure 3: The system view of the modifiability PRM.

Figure 4: The modifiability PRM containing classes and attributes, including numbering of the attributes causal dependencies.

For the interested reader (Lagerström 2007) presents an early version of the modifiability metamodel.
5. Modeling

This section presents the approach of instantiating the modifiability metamodel, as well as the possible benefits gained when it is employed in change projects.

5.1. Metamodel instantiation

The purpose of the modifiability metamodel, described in section 4, is for it to be instantiated in software change projects in order to estimate the project cost (in man-hours) and to highlight risks, thus supporting decision making and planning.

The process of instantiating the metamodel basically contains two steps: data collection and modeling. Based on the metamodel elements the modeler gets input to what data she should collect for her models. The modifiability metamodel, cf. Fig. 4, contains information regarding what classes to instantiate into objects, how these relate to each other, and what attributes they need for change cost analysis.

In project M studied at a large Nordic transportation company (described in section 6.4) data was collected by interviewing and surveying people involved in the project, and by studying project documentation. Additional data was also collected with the development tools used at the company, for instance the size and component internal coupling measures. Based on the data collection an architecture model was instantiated for the project, the main view is illustrated in Fig. 10. The instantiated model view presented illustrates project M on a high level, i.e. excluding the underlying classes and their attributes.

According to the metamodel the instantiated model should for instance contain systems and how these relate to each other. In project M there were seven systems involved in the change: ticket delivery, youth travel, webshop, sales, content management, seat booking, and journey planner. Each system has, according to the metamodel, the attributes understandability, internal coupling, size, and complexity. The collection of systems also has the attribute external coupling (calculated based on the information in the instantiated model). Fig. 9 illustrates the external coupling view of the instantiated architecture model for project M.

The attributes of the metamodel are all defined with the scales described in (Lagerström et al. 2009b). Excerpts of four attribute scales are presented in Table 2. Using a fine grained scale provides clear definitions and allows for statistical analysis. These scales are used when data is collected for the instantiation of the metamodel. However, since the approach is formalized with probabilistic relational models, the input for the instantiated architectural model needs to be on a discrete scale with a finite number of states. Also, the method used for defining the CPDs of the metamodel restricts the attribute scales to be discrete with three defined states, cf. section 3.2. Therefore, the data collected is transformed according to the rules described in Table 2.
Table 2: Attribute definitions, data collection scales, transformation rules, and model calculation scales for the class system.

<table>
<thead>
<tr>
<th>Class</th>
<th>Attribute</th>
<th>Definition</th>
<th>Data collection scale</th>
<th>Transformation rule</th>
<th>Model calculation scale</th>
</tr>
</thead>
</table>
|       | Understandability  | Time, in percentage compared to total change activity time, spent on trying to understand the system. | [0, ..., 100]         | 41-100 % = Difficult
             |                    |                                                                           | 21-40 % = Normal
             |                    |                                                                           | 0-20 % = Easy          | (Difficult, Normal, Easy) |
|       | Internal coupling  | Number of actual relations between the components in this system divided with the number of possible relations. | [0, ..., 100]         | 41-100 % = Tight
             |                    |                                                                           | 16-40 % = Normal       | (Tight, Normal, Loose)   |
|       | Size               | Total number of components in the system                                  | (0, 1, 2, ..., ∞)     | 500 - ∞ = Large
             |                    |                                                                           | 40 - 499 = Medium      | (Large, Medium, Small)   |
| System | Complexity         | Subjectively evaluated. “How complex is the system?”                     | (Complex, Medium, Not complex) |                           | (Complex, Medium, Not complex) |

5.2. Probabilities estimation vs. cost in man-hours

The PRM based metamodel presented handles all attribute values as probabilities, including the goal attribute Cost. This means that change cost has probability values being either High, Medium or Low. To enable estimations in man-hours these probabilities need to be translated into man-hours. In this paper the probability transformation is approached by the introduction of segments, cf. Table 3. These segments take into consideration whether the projects are considered being of either Large, Medium or Small size.

To be able to utilize the estimation capabilities of the metamodel for software change project cost analysis within the accuracy ranges in Table 6 (further described in section 7.3), a subjective prediction of the size of the project needs to be done. This segmentation of project size is preferably performed by project managers having experience of the project culture in the company where the architectural models are being applied.

Here a simple categorization aiming at determining if, dependent on e.g. company culture, the largest projects usually turns out to demand around 40 000, 12 000 or 5 000 man-hours. Some companies always perform their work in small projects that in the worst cases means around 5 000 man-hours. Other companies work in large projects costing approx. 40 000 man-hours. Thus, when using the modifiability metamodel the modeler (e.g. project manager) must make a decision whether the project he or she is modeling is a large, medium, or small project. For a large project this means that the probabilities of the project being High, Medium, or Low in the metamodel are mapped to the man-hour levels 40 000, 6 000 and 1 000 respectively. This has so far only been done a posteriori and further research is being conducted to classify these more objectively and generally. Once a project has been fitted into a segment, then it is possible to make a cost estimation in man-hours with a reasonable accuracy. Table 3 depicts the transformation
from probabilities of High, Medium, and Low cost to number of man-hours for Large, Medium, and Small projects.

<table>
<thead>
<tr>
<th>Segment</th>
<th>High</th>
<th>Medium</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large</td>
<td>40 000</td>
<td>6 000</td>
<td>1 000</td>
</tr>
<tr>
<td>Medium</td>
<td>12 000</td>
<td>6 000</td>
<td>1 000</td>
</tr>
<tr>
<td>Small</td>
<td>5 000</td>
<td>2 500</td>
<td>1 000</td>
</tr>
</tbody>
</table>

Table 3: Cost intervals for categorization of change project size.

The cost in man-hours, $C$, is calculated as an expectation value $E(C)$.

$$E(C) = \sum_{i=L}^{H} U_i \cdot P(C = i),$$

where $U_i$ is the utility value of the attribute Cost given that the Cost equals $i$.

In Fig. 10 we can see that the attribute Cost (prob.) is High with a 26 % probability, Medium with a 33 % probability, and Low with a 41 % probability. Since project M is considered to belong to segment Medium, Table 3 gives us the utility values $U_{\text{High}} = 12000$, $U_{\text{Medium}} = 6000$, and $U_{\text{Low}} = 1000$.

Thus the expectation value for the cost in project M is

$$E(C) = 12000 \cdot 0.26 + 6000 \cdot 0.33 + 1000 \cdot 0.41 = 5510.$$

5.3. Modeling benefits

A decision maker, typically project managers working with software change, will be provided with four sorts of valuable information when utilizing the modifiability metamodel. Firstly, the expected costs of a set of change projects can be estimated providing a cost ranking of the projects in the company portfolio. This means that a more rational decision making is enabled.

Secondly, it is possible for a decision maker to test different scenarios regarding the objects and attributes in the instantiated architectural model in order to try to lower the cost of a specific project and thereby providing increased decision support for software project portfolio prioritization. This could for example be realized by involving other developers having more suitable experience within the chosen design specific language in the project.

Thirdly, when a project has been chosen from the portfolio to be initiated the architectural model will be able to aid the project manager in the planning phase of the project. The manager can, for instance, get suggestion on architectural and
development team size, as well as elaborate how the team expertise will affect the outcome.

Fourthly, the instantiated architectural model can be used for conducting risk analysis. The model is able to expose parts of the project with a high risk of increasing its cost. Hence, action plans can be set up accordingly to mitigate the occurrence of these risks. This enables the resources chosen for the project to be optimized for the project specific change activities. Hence, the risk of the project exceeding its given budget and timeframes are somewhat more controllable.

6. Architectural models and analysis

Several case studies have been conducted based on the modifiability meta-model and the enterprise architecture analysis approach presented. In all, four multiple case studies have been conducted and within these 21 software change projects have been studied. In the first case two projects within a Nordic consultancy firm were studied (projects A and B). The second case considered eight projects conducted within a large Nordic manufacturing company (projects C to J). Case three contained two projects at a leading Nordic software and hardware vendor (projects K and L). In the fourth case nine change projects where studied at a Nordic transportation company (projects M to U).

This section focuses on presenting the companies involved in the four multiple case studies. Four projects are presented in text and with instantiated architectural models, one from each case study. The actual costs and the estimated values of all 21 software change projects are presented in section 7.

6.1. Case study 1 – Nordic consultancy firm

This consultancy firm, established in 1993, is one of the leading Nordic consulting firms in the field of Product Data Management (PDM) and related software technology areas such as Product Lifecycle Management (PLM), Product Engineering Systems, Product Database Applications, and Document Management.

Two projects (called project A and B) were studied at the consultancy firm. Both projects were large software change projects with the aim of implementing a number of end-user requirements in several systems. The projects were implemented at a large Nordic IT-provider and the development was done by a subcontractor. Project A is further described in the next paragraph and Fig. 5 presents an instantiated architectural model view focusing on the project and change activity classes.

Project A ended up costing approximately 20 000 man-hours, where 6 800 man-hours were used for the architectural changes and 13 200 man-hours for the component changes. The largest costs were the development cost and test cost for
the component changes. The smallest cost was the learning cost for the architect-
tural changes. Comparing the project outcome with the estimation provided by
the instantiated model, cf. Fig. 5, we can see that the accuracy of the estimated
value is as high as 99 %. We can also see in the model that the technical changes
of the components are both difficult and large, thus it was no surprise that the
largest costs were the development and test costs for the component changes.

\[
\begin{align*}
\text{Cost (man-hours)} &= 19,860 \\
\text{Cost (prob.)} &= \text{High (44), Medium (34), Low (22)} \\
\text{Change project: A} \\
\text{Architectural change activities: A} \\
&\quad \text{-Cost = High (35), Medium (25), Low (40)} \\
&\quad \text{-Synchronization need = Low} \\
\text{Component change activities: A} \\
&\quad \text{-Cost = High (54), Medium (33), Low (13)} \\
&\quad \text{-Synchronization need = High} \\
\text{Techn. changes to arch: A} \\
&\quad \text{-Change difficulty = Difficult} \\
&\quad \text{-Change size = 5 (small)} \\
\text{Techn. changes to comp: A} \\
&\quad \text{-Change difficulty = Difficult} \\
&\quad \text{-Change size = 1,000,000 (large)}
\end{align*}
\]

Figure 5: An excerpt of the instantiated PRM for project A presenting the classes: change project, change activities, and technical changes.

6.2. Case study 2 – Nordic manufacturing company

The second case study was conducted at a large Nordic manufacturing company.
The company is active in some 100 countries and has over 30,000 employ-
ees. More than 2,000 people work with development and research, mainly within
the Nordic region. The production facilities are spread out in Europe and Latin
America.

In this multiple case study eight projects (labeled project C to J) were stud-
ied. We will focus on presenting some information regarding Project F. Project
F was initiated since the company predicted increased sales and thus an increased
amount of employed products. To still be able to manage the products a new func-
tion in the product management portal was needed. In addition to this there was
a need for a more secure, redundant and scalable server environment to be set up
along the development of the new software application. The project had several
objectives concerning improvement of the product management business, namely;
to reduce the amount of hours spent on administration related to product manage-
ment, to improve the support for the distributors of the products, to improve the
quality of the already present services, to obtain a scalable, redundant and secure
communication infrastructure and finally, to future proof the communication and server environment in terms of more long lasting functionality.

The main risks found in project F, cf. Fig. 6, were: the expertise of the developer team was considered to be low since the people involved only had 0-1 years of experience with change work and with the programming language used. Also, the architect team, which has high expertise, only spend 0-20 % of their time on this particular project. The change management process is measured to be only at maturity level 1 (scale: level 0-5). Furthermore, the external coupling of the systems involved were considered to be high.

Project F ended up costing approximately 20 000 man-hours and the modifiability metamodel estimated the cost to 15 230 man-hours. Thus, the estimation accuracy for this project is 76 %.

![Figure 6: The main view of the instantiated PRM for project F of the multiple case study at a Nordic manufacturing company.](image)

6.3. Case study 3 – Nordic software and hardware vendor

The third case study was conducted at a large Nordic software and hardware vendor. The company is a global leader in its domain. The company has over 100,000 employees worldwide and they have customers and local distributors in over 100 countries. The case study was carried out at one of the development departments at the company. The department’s work mainly focus on management
and further development of an information and control system. This system is sold as a standard product, however each implementation is modified in order to match the unique requirements of the customer.

The projects studied in this case were two projects (named project K and L) in a large program consisting of numerous projects. The projects, although part of a program, were treated by both the customer and the vendor as separate projects. The main program’s purpose was to implement a large upgrade of a National Control Center in a large non-European country. The upgrade consisted of a large amount of additional functionality that was supposed to be added to a standard product delivered by the vendor. In the project presented here, from now on called Project L, the main task was to add a new component to the standard product. The main function of this new component is to provide a tool for operators to evaluate and analyze optional transactions with other companies. The new component also required integration with several other already existing components of the system.

As can be seen in the instantiated organizational view of project L, cf. Fig. 7, there were six persons involved in project L. Five developers and one architect. Since many of the developers only had 0-2 years of experience for several of the experience attributes, the average expertise level of the developer team was assessed to be Low.

![Figure 7: An organizational view of the instantiated PRM for project L.](image)

Project L was estimated to 800 man-hours by the project manager at the company. His estimate was based on his experience without any tool support. As can be seen in Fig. 8, the instantiated model of project M provided an estimate of 3085 man-hours. The actual cost in man-hours was approximately 3 200. In this
case the expert estimate had an accuracy of 25%, while the metamodel provided an estimate with a 96% accuracy.

Figure 8: The main view of the instantiated PRM for project L of the multiple case study at a Nordic software and hardware vendor.

6.4. Case study 4 – Nordic transportation service company

The fourth case study was conducted at a large Nordic transportation company. The company provides transportation services mostly within one of the Nordic countries, but also has some services in neighboring countries. They carry hundreds of millions of passengers every year, mainly with train services, and employ several thousand people.

In total, nine software change projects (project M to U) were studied at the transportation company. We here focus on presenting one project, from now on called *Project M*. Project M was a project focusing on modifying the company’s public ticket webshop into an almost completely new version of it. The focus was on adding and changing functionality in the webshop software, making it possible for the passengers to buy tickets online easier, faster, more secure and more reliable. One especially important part of the project was to increase the usability with focus on the user interface for finding journeys, choosing ticket type, viewing price information, and pdf-ticket printing. The project included developing a new
The webshop system is integrated with a sales system. Also, the webshop system and the sales system are integrated with a seat booking system. Furthermore, the sales system and the seat booking system are integrated with a journey planning system. In all, there are seven systems, coupled together, involved in the change project, i.e. seven systems in need of modifications. In Fig. 9 an external coupling view of the instantiated architecture model presents the systems in project M and how these communicate.

The instantiated model presented in Fig. 10 presents project M on a high level, i.e. excluding the aggregating classes and their attributes. As can be seen in this figure the estimated cost was 5 510 man-hours and the actual cost was 5 810 man-hours. The metamodel provided an estimation with a 95 % accuracy in this case.

7. Validation

The elements and structure of the modifiability metamodel as well as its estimation capabilities need to be evaluated and validated.
While the use of academic papers reflecting research serves as a good foundation for the metamodel creation, it is not completely trustworthy. There are several reasons for this. First, the scientific literature is not complete, so when creating a metamodel, it might be necessary to fill in some blanks with hypotheses unsupported by the literature. Second, the scientific literature is not always coherent, so when creating a metamodel it might be unavoidable to make more or less controversial choices. Third, the modeler might be biased and thus involuntarily introduce distortions. Expert validation of the metamodel attributes serves as a good function in minimizing these uncertainties.

Since the attribute PRMs of the metamodel are defined based on expert knowledge mapped to a three graded scale, there is a risk that the estimations are less accurate than wanted. Data based on 21 software change projects is used for testing the estimation capabilities of the metamodel.

The three most important questions are: 1) Are there attributes missing in the metamodel that should be added? 2) Are there superfluous attributes in the metamodel that could be removed? 3) Does the metamodel provide good estimation capabilities?

Together, the first and second question determines whether the metamodel contains the appropriate elements, and will be addressed in subsections 7.1 and
The third question concerns the whole metamodel, both the qualitative and quantitative structure i.e. both the classes, reference slots, attributes, and causal dependencies, as well as the conditional probability distributions and aggregate functions. The estimation capability is validated by studying 21 change projects and by comparing the estimated cost with the actual cost outcome of the projects. This is addressed in subsection 7.3.

7.1. Are there attributes missing in metamodel?

The first question, “are there attributes missing in the metamodel?”, was posed to a number of respondents, both academics in an online survey and industrial people during workshops. Two workshops with industrial respondents, with 8 and 39 respondents respectively, were carried out. One online survey was sent out to academics in the field of software modifiability, this survey had 50 respondents. Thus, the total number of respondents was 97.

In the workshops and surveys the respondents were also provided with questions regarding their qualification as experts. 14 persons were excluded from the two workshops and ten persons were excluded from the survey. Either these respondents had too little experience (less than three years) or they themselves said that they did not feel certain at all about their answers. This resulted in 33 workshop experts, where 17 provided suggestions on attributes that could be missing in the metamodel. Of the 40 survey experts 13 provided suggestions on missing attributes. Thus, in total the number of experts with suggestions on missing attributes was 30.

The only attributes that two or more experts were missing are presented in Table 4. The attributes with the most votes in our survey and workshops had 3 experts out of the total 73 missing them, which is only 4.1%. If we consider the possibility that the 43 persons not missing any attributes at all skipped the open-ended questions due to time restrictions and instead restrict attention to the group that suggested new attributes we have 3 votes out of a total 30 (10%). Since there is no strong agreement among the 30 experts on which attributes are missing, the indication is that the metamodel contains an appropriate set of attributes. Thus, the answer to question one is: no, the modifiability metamodel does not seem to be missing any attributes. Nevertheless, the attributes suggested are interesting and should be explored in future case studies and surveys.

7.2. Does the metamodel contain attributes that can be removed?

The second question, ”are there superfluous attributes in the metamodel that could be removed?”, was addressed by analyzing the strength of the causal dependencies between the metamodel attributes, as elicited from the experts. That is, if experts find that there is a strong causal connection between two attributes,
<table>
<thead>
<tr>
<th>Class</th>
<th>Attribute</th>
<th>No of answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management</td>
<td>Degree of support</td>
<td>3</td>
</tr>
<tr>
<td>Testing process</td>
<td>Test coverage</td>
<td>2</td>
</tr>
<tr>
<td>System</td>
<td>Level of quality goals restrictions</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Platform independence</td>
<td>2</td>
</tr>
<tr>
<td>Requirements</td>
<td>Number of authors</td>
<td>3</td>
</tr>
<tr>
<td>specification</td>
<td>Number of changes during project</td>
<td>2</td>
</tr>
<tr>
<td>Architecture goals</td>
<td>Prioritized and communicated</td>
<td>3</td>
</tr>
<tr>
<td>Business organization</td>
<td>Stability</td>
<td>2</td>
</tr>
<tr>
<td>Change organization</td>
<td>Geographic distance (culture &amp; language diff.)</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Understandability of business objects</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Use of a common information model</td>
<td>3</td>
</tr>
<tr>
<td>Change activities</td>
<td>Time restrictions (deadline)</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 4: The workshop and survey results for the question “Which attributes are missing in the metamodel?”.

then these attributes should be present in the metamodel. Conversely, if experts find that there is no causal connection between two attributes, then the parent attribute could be removed from the metamodel. Table 5 summarizes the causal dependency strengths found.

There were in total 99 respondents answering the questionnaire. 13 were industrial people providing answers at two workshops and 86 were academics providing their answers via two online surveys.

The question asked to the respondents both in the workshops and surveys was: “How large is the effect presented in the following statements?” The respondents were provided with statements each corresponding to a causal dependency in the metamodel. See Fig. 4 for all corresponding causal relationships. The statements were all arranged as the following examples; “Change management process maturity affecting component documentation”, which corresponds to the relationship labeled as number 32 in the metamodel. “Developer team expertise affecting the component change activities cost”, corresponding to the causal dependency labeled as number 19. The answer alternatives were given to the experts on the following scale; High effect, Low effect, No effect, or I don’t know.

High effect between two attributes means that in most cases when you change the value of the parent attribute the child attribute changes as well. That is, the
attributes are causally dependent. *Low effect* between two attributes means that in some cases when you change the value of the parent attribute the child attribute changes as well. I.e. there is a low causal dependency between the attributes. *No effect* between two attributes means that in no (or very few) cases when you change the value of the parent attribute the child attribute changes as well. That is, there is no causal dependency between the attributes. The *I don’t know* answer means that either the respondents did not understand the concepts in the question or the respondents understood the concepts but did not have the experience to estimate the actual effect.

In the workshops and surveys the persons were also provided with questions regarding their qualification as experts. In the workshops, two persons were excluded due to lack of expertise; both persons stated that they had not enough experience in the field. Fourteen persons were excluded from the surveys, either they had too little experience (less than three years), they themselves said that they did not feel certain at all about their answers, or the answer *I don’t know* was given to more than 50 % of the questions asked.

<table>
<thead>
<tr>
<th>Relation</th>
<th>High effect</th>
<th>Low effect</th>
<th>No effect</th>
<th>I don’t know</th>
<th>No of answers</th>
<th>No eff. percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td>19</td>
<td>2</td>
<td>6</td>
<td>52</td>
<td>4.5%</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>26</td>
<td>7</td>
<td>4</td>
<td>52</td>
<td>17.1%</td>
</tr>
<tr>
<td>3</td>
<td>44</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>52</td>
<td>0.0%</td>
</tr>
<tr>
<td>4</td>
<td>22</td>
<td>24</td>
<td>2</td>
<td>4</td>
<td>52</td>
<td>4.3%</td>
</tr>
<tr>
<td>5</td>
<td>44</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>52</td>
<td>2.0%</td>
</tr>
<tr>
<td>6</td>
<td>28</td>
<td>19</td>
<td>0</td>
<td>5</td>
<td>52</td>
<td>0.0%</td>
</tr>
<tr>
<td>7</td>
<td>29</td>
<td>17</td>
<td>1</td>
<td>5</td>
<td>52</td>
<td>2.2%</td>
</tr>
<tr>
<td>8</td>
<td>27</td>
<td>18</td>
<td>2</td>
<td>5</td>
<td>52</td>
<td>4.4%</td>
</tr>
<tr>
<td>9</td>
<td>43</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>52</td>
<td>2.0%</td>
</tr>
<tr>
<td>10</td>
<td>37</td>
<td>13</td>
<td>1</td>
<td>1</td>
<td>52</td>
<td>2.0%</td>
</tr>
<tr>
<td>11</td>
<td>33</td>
<td>14</td>
<td>3</td>
<td>2</td>
<td>52</td>
<td>6.4%</td>
</tr>
<tr>
<td>12</td>
<td>41</td>
<td>9</td>
<td>1</td>
<td>1</td>
<td>52</td>
<td>2.0%</td>
</tr>
<tr>
<td>13</td>
<td>44</td>
<td>6</td>
<td>0</td>
<td>2</td>
<td>52</td>
<td>0.0%</td>
</tr>
<tr>
<td>14</td>
<td>28</td>
<td>20</td>
<td>1</td>
<td>3</td>
<td>52</td>
<td>2.1%</td>
</tr>
<tr>
<td>15</td>
<td>36</td>
<td>14</td>
<td>1</td>
<td>1</td>
<td>52</td>
<td>2.0%</td>
</tr>
<tr>
<td>16</td>
<td>27</td>
<td>18</td>
<td>3</td>
<td>4</td>
<td>52</td>
<td>6.7%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Relation</th>
<th>High effect</th>
<th>Low effect</th>
<th>No effect</th>
<th>I don’t know</th>
<th>No of answers</th>
<th>No eff. percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>23</td>
<td>13</td>
<td>2</td>
<td>4</td>
<td>42</td>
<td>5.6%</td>
</tr>
<tr>
<td>18</td>
<td>14</td>
<td>21</td>
<td>1</td>
<td>6</td>
<td>42</td>
<td>2.9%</td>
</tr>
<tr>
<td>19</td>
<td>38</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>42</td>
<td>0.0%</td>
</tr>
<tr>
<td>20</td>
<td>18</td>
<td>8</td>
<td>2</td>
<td>14</td>
<td>42</td>
<td>7.7%</td>
</tr>
<tr>
<td>21</td>
<td>35</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>42</td>
<td>0.0%</td>
</tr>
<tr>
<td>22</td>
<td>36</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>42</td>
<td>2.4%</td>
</tr>
<tr>
<td>23</td>
<td>10</td>
<td>25</td>
<td>4</td>
<td>3</td>
<td>42</td>
<td>11.4%</td>
</tr>
<tr>
<td>24</td>
<td>23</td>
<td>16</td>
<td>1</td>
<td>2</td>
<td>42</td>
<td>2.6%</td>
</tr>
<tr>
<td>25</td>
<td>23</td>
<td>15</td>
<td>1</td>
<td>3</td>
<td>42</td>
<td>2.6%</td>
</tr>
<tr>
<td>26</td>
<td>29</td>
<td>12</td>
<td>1</td>
<td>0</td>
<td>42</td>
<td>2.4%</td>
</tr>
<tr>
<td>27</td>
<td>30</td>
<td>11</td>
<td>1</td>
<td>0</td>
<td>42</td>
<td>2.4%</td>
</tr>
<tr>
<td>28</td>
<td>27</td>
<td>12</td>
<td>1</td>
<td>2</td>
<td>42</td>
<td>2.6%</td>
</tr>
<tr>
<td>29</td>
<td>39</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>42</td>
<td>0.0%</td>
</tr>
<tr>
<td>30</td>
<td>29</td>
<td>10</td>
<td>1</td>
<td>2</td>
<td>42</td>
<td>2.6%</td>
</tr>
<tr>
<td>31</td>
<td>31</td>
<td>8</td>
<td>0</td>
<td>3</td>
<td>42</td>
<td>0.0%</td>
</tr>
<tr>
<td>32</td>
<td>18</td>
<td>15</td>
<td>3</td>
<td>6</td>
<td>42</td>
<td>9.1%</td>
</tr>
</tbody>
</table>

Table 5: Survey and workshop data regarding the strength of influence between causally related attributes in the metamodel. The data is also used for defining the CPDs as explained in section 3.2.

Since the attributes in general have either *high effect* or *high/low effect* in relation to their parents, whereas very few had *no effect*, the attributes in the metamodel all seem useful for modifiability analysis. As can be seen in Table 5, no causal dependency has more than 17.1% answers on *No effect*. We interpret these
low percentages to indicate that there are no attributes in the metamodel with no effect on its causally related attributes and by that the cost of making changes. Thus the answer to question two is: no, the modifiability metamodel does not seem to have any superfluous attributes that could be removed. However, some causal dependencies in the metamodel do have some answers on Low effect. These dependencies will be further validated in future change projects and questionnaires. Possibly one or two attributes can be removed or replaced, but this requires more research.

7.3. Does the metamodel provide good estimation capabilities?

The third question, “does the metamodel provide good estimation capabilities?”, was addressed by conducting four different multiple case studies. In total 21 software change projects were modeled. Section 6 presents the four companies and four selected projects, one from each case study.

Since all studied projects are completed, data concerning their actual costs is obtainable. The estimated cost for each of the 21 projects is listed and compared to the actual cost of each project, cf. Table 6. The last column indicates the accuracy of the estimation, i.e. how close to the real value the estimation is. The accuracy is calculated using the magnitude of the relative error (MRE), as defined by (Conte et al. 1985). Suppose $E$ is the estimate of a value and $A$ is the actual value, then the magnitude of the relative error for the estimate is

$$MRE = \frac{|A - E|}{A}.$$  

Accuracy is calculated as $1 - MRE$. This can be seen as the primary measure for the quality of the estimation the modifiability metamodel is capable of. E.g. for project M we can see that the actual cost turned out to be 5 810 man-hours. The model estimated the cost to be 5 510 man-hours. Thus, the accuracy of the estimation is calculated to be 95%.

Studying Table 6 we can see that the smallest projects, ranging from 262 to 1200 man-hours, are the most difficult ones to estimate with the modifiability metamodel. This outcome, i.e. that the estimation capabilities of the metamodel would not work that well for the smallest software projects, was expected. For instance, projects with few persons and components involved do not need documentation and process support to the same extent as large projects do. For the smallest projects the metamodel would probably contain a different set of classes and attributes focusing on more detailed development issues. Also, the estimation need for really small projects might be superfluous since the estimation work might consume too much time in relation to the business value it provides. Thus, one issue with the metamodel estimation concerns the project size: at what size does the metamodel no longer provide an acceptable estimation accuracy? The
Table 6: Studied projects with the actual costs, estimated costs, and accuracy of all conducted estimations.

Data from the 21 studied change projects provides an indication that the metamodl should not be employed in change projects under 1 200 man-hours. There are other more suitable methods and models for the smallest projects, e.g. planning poker as discussed in section 8.3.

A question raised regarding the project size is: how can one know which projects that are large enough for the modifiability metamodel in advance? This was analyzed and discussed during the fourth case study. In the nine projects studied at the transportation company (projects M–U) there was a clear distinction between the projects with acceptable accuracies and the ones being poorly estimated by the metamodel. Two variables available at project start that differed between these projects were found, these are **Number of systems involved in the change** and **System change distribution**. As we can see in Table 7 projects M, R and U are successfully estimated, while O, N, Q and S are poorly estimated. Projects P and T are borderline cases. In the successfully estimated projects there were between five and seven systems involved in the change and the change distribution was high/medium between the involved systems. In the less successful cases the number of systems involved were two–four and the change distribution was low.

Thus, in projects with many systems being changed and where the changes are...
spread out between these systems the modifiability metamodel seems to produce more accurate estimates than in the case when there are few systems involved and when the changes are more focused to few systems. As a consequence of this, the answer to the posed question is: projects aiming to modify five or more systems and with a high change distribution between these systems seem to be the ones benefiting from employing the proposed modifiability metamodel.

<table>
<thead>
<tr>
<th>Project</th>
<th>Actual cost (man-hours)</th>
<th>Estimated cost (man-hours)</th>
<th>Accuracy</th>
<th>Number of systems involved</th>
<th>System change distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>5 810</td>
<td>5 510</td>
<td>0.95</td>
<td>7</td>
<td>High</td>
</tr>
<tr>
<td>R</td>
<td>2 054</td>
<td>1 915</td>
<td>0.93</td>
<td>5</td>
<td>High</td>
</tr>
<tr>
<td>U</td>
<td>3 595</td>
<td>4 110</td>
<td>0.86</td>
<td>7</td>
<td>Middle</td>
</tr>
<tr>
<td>P</td>
<td>4 458</td>
<td>6 380</td>
<td>0.57</td>
<td>7</td>
<td>High</td>
</tr>
<tr>
<td>T</td>
<td>1 082</td>
<td>1 680</td>
<td>0.45</td>
<td>3</td>
<td>Low</td>
</tr>
<tr>
<td>O</td>
<td>952</td>
<td>2 040</td>
<td>&lt;0</td>
<td>3</td>
<td>Low</td>
</tr>
<tr>
<td>N</td>
<td>894</td>
<td>2 295</td>
<td>&lt;0</td>
<td>4</td>
<td>Low</td>
</tr>
<tr>
<td>Q</td>
<td>454</td>
<td>2 005</td>
<td>&lt;0</td>
<td>2</td>
<td>Low</td>
</tr>
<tr>
<td>S</td>
<td>262</td>
<td>1 805</td>
<td>&lt;0</td>
<td>3</td>
<td>Low</td>
</tr>
</tbody>
</table>

Table 7: Studied projects in the fourth case study conducted at a large Nordic transportation company.

According to Conte et al. an acceptable level for mean accuracy is something higher than or equal to 75 %. Calculating the mean accuracy for the 15 projects above 2 000 man-hours results in a 88 % accuracy. Conte et al. use this mean accuracy notion to define a measure of prediction quality (\(PRED(q)\)). In a set of \(n\) projects, let \(k\) be the number of projects where the mean accuracy is higher than or equal to \(q\). Then

\[ PRED(q) = \frac{k}{n}. \]

According to Conte et al. an estimation technique is acceptable if \(PRED(0.75) = 0.75\). This means that in 75 % of the time the estimated values falls within 75 % of their actual values. In our case \(PRED(0.75) = 0.87\), thus the metamodel is acceptable as an estimation technique. However, if we include the six projects of 1 200 man-hours or less then \(PRED(0.75) = 0.62\).

Thus the answer to question three is: yes, the modifiability metamodel provides good estimation capabilities (at least for software change projects over 2 000 man-hours). However, since the smallest projects (the ones of 1 200 man-hours and less) seems to be more difficult for the modifiability metamodel to estimate, this will be addressed separately in future research. Another topic for further research related to the metamodel’s estimation capability is the size segmentation.
discussed in section 5.2. Also, future research will include studying more change projects from start to end to provide another level of validation.

8. Comparison with other models and methods

Since Enterprise Architecture is a discipline on the rise there are no alternatives within this modeling field that can be used for comparison. There are however other disciplines that have addressed the cost estimation problem. This section presents three of such alternative models and methods for cost estimation. These are briefly presented and their estimation capabilities are compared with the ones of the modifiability metamodel. Section 8.1 presents COCOMO II, section 8.2 presents function points, and section 8.3 presents planning poker.

8.1. COCOMO II

COCOMO, COnstructive COSt MOdel, was in its first version released in the early 1980’s. It became one of the most frequently used and most appreciated software cost estimation models of that time. Since then, development and modifications of COCOMO has been performed several times to keep the model up to date with the continuously evolving software development trends. The latest version of COCOMO, called COCOMO II, had its estimation capabilities calibrated in the year 2000 with the help of information from 161 project data points and 8 experts. This latest calibrated version of COCOMO II uses the probabilistic Bayesian approach for turning a priori obtainable data into estimates of costs related to an a posteriori state of a software development or modification project (Chulani et al. 1999; Boehm et al. 2000).

Table 8 contains a comparison between COCOMO II and the modifiability metamodel. The COCOMO II.2000 calibration has an accuracy of 75 % of the actual cost 68 % of the time before stratification and 76 % of the time after stratification. This is a major improvement compared to the performance of the COCOMO II.1997 version which had an accuracy of 75 % of the actual cost 49 % of the time before stratification and 55 % of the time after stratification (Boehm et al. 2000). The COCOMO stratification is done by calibrating the model’s multiplicative coefficient to each of their major sources of project data (Chulani et al. 1999). (Chulani et al. 1999) recommend organizations using COCOMO II.2000 to calibrate it using company specific data to increase the model accuracy. The modifiability metamodel on the other hand has an accuracy of 75 % of the actual cost in 62 % of the time before and in 87 % of the time after its removal of the projects of 1 200 man-hours or less.

Based on this comparison the estimation capabilities of the modifiability metamodel (based on all projects) are similar to the capabilities of COCOMO II before their stratification. It is slightly better than the 1997 version and somewhat less
accurate than the 2000 version. However, when focusing on the 15 projects of 2 000 man-hours and above the modifiability metamodel seems to provide a 75 % accuracy for more projects than COCOMO II. Before any conclusions can be drawn based on this there are still some issues that need to be addressed. Firstly, the stratification and the project exclusion are not the same, thus maybe not really comparable. Secondly, the COCOMO results are tested with 161 projects while the modifiability metamodel results are tested with 21 (15) projects. Hence, the COCOMO II results are probably more stable. Although there are some issues left to be solved our results show great promise for future research and use of the enterprise architecture approach and the modifiability metamodel.

8.2. Function points

The function points analysis is a method of quantifying the size and complexity of a software system in terms of the functions that the system delivers to the user. Counting function points is done by considering the linear combination of five basic software components (inputs, outputs, master files, interfaces, and inquiries), each at one of three levels: low, average or high. This count is called unadjusted function points (UFP). The final number of a software system’s function points is arrived by multiplying the UFP by an adjustment factor that is determined by considering 14 aspects of processing complexity (Matson et al. 1994).

(Matson et al. 1994) did a major study of function point’s ability to estimate software cost in 1994. They analyzed function point data from 104 projects obtained from a major corporation. In the study two function point models were tested, one logarithmic transformation model (Model A) and one multiple regression model (Model B). Model A had an accuracy of 75 % in 64 % of the cases and model B in 68 % of the cases, cf. Table 8. While the modifiability metamodel has a 75 % accuracy in 62 % of the cases before removing the projects of 1 200 man-hours or less and 87 % after. Thus, both function point models perform slightly better than the modifiability metamodel before project removal. However, when the smallest projects are removed the metamodel seems to perform better.

8.3. Planning poker

Planning poker is a so called expert estimation method especially useful in agile software projects. The idea is to divide a project into smaller functional parts, which are described as short stories. For each described story the team members in the project are supposed to write down an estimate on a card, without any discussion among the team members. The estimates for each story are then discussed (if different) and new estimates are written down. This iterative process continues until the team has written similar estimations for all stories.
Table 8: Comparing the estimation accuracy of the modifiability metamodel, COCOMO II, and function points.

(Moløkken-Østvold et al. 2008) presents a study of using planning poker for combining expert estimates in software projects. In their study of one software project containing 55 stories they found that the mean estimation accuracy was 82 %, cf. Table 9. While, the mean estimation accuracy of the modifiability metamodel was 88 % (when studying the 15 project above 2 000 man-hours). As we can see they provide rather similar estimation accuracies.

The project studied by (Moløkken-Østvold et al. 2008) ended up in approximately 430 man-hours and had the mean accuracy of 82 %. Thus, planning poker seems to be a good complement to the modifiability metamodel for the projects of 1 200 man-hours and less. It is however unclear how appropriate planning poker is for large projects.

Table 9: Comparing the mean estimation accuracy of the modifiability metamodel and planning poker.

<table>
<thead>
<tr>
<th>Model / method</th>
<th>Mean accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>The modifiability metamodel, all above 2 000 man-hours</td>
<td>88%</td>
</tr>
<tr>
<td>Planning poker</td>
<td>82%</td>
</tr>
</tbody>
</table>

9. Discussion and research outlook

During the research work, several issues concerning the validity have been addressed. Multiple sources in both the development phase and the validation phase were used. Key stakeholders have been addressed for reviews of drafts. A chain of evidence was established. Theory was developed and used. There are however some threats to the validity. These threats mainly concern the 21 change projects studied in the validation phase when focusing on the estimation capabilities. Since there was no possibility of studying projects from start to end
most data gathered in these case studies was collected after the projects were finished. Thus providing a final cost of the projects for comparison, but also inferring an uncertainty of the a priori value of the data. Another issue being a threat to the validity concerns the scales used when collecting and analyzing data, especially the transformation between these scales. The cost interval segmentation used when estimating the costs is also a validity issue since this so far only has been done after the projects were finished.

The quantitative part of the metamodel was created by surveying modifiability experts both from industry and academia, in total 138 people were surveyed. Of these, 110 were classified as modifiability experts as explained in section 7. The industry participants were surveyed during three separate workshops; one at a company working with enterprise software system change, one at the Software Metrics network of the Swedish software and computer association and one at the second annual Enterprise Architecture Symposium at the Royal Institute of Technology. The academic participants were chosen based on their publications at the 11th and 12th European Conference on Software Maintenance and Reengineering. Although the participants have been chosen based on their expertise in the field of enterprise system modifiability and that both industry practitioners and academics are present in the survey, it is difficult to be absolutely sure that the sample is representative. We do however believe that the metamodel foundation with academic literature, the survey data and the case studies provide results sound enough. This can nevertheless be further addressed in future research.

Reliability was achieved by documenting the research work during all phases. All case studies have their own reports describing data collection and analysis. The aim has been to operationalize as many steps as possible, e.g. conducting surveys with predefined choices and using methods with well defined guidelines and rules, thereby allowing other investigators to repeat the work.

To the industrial practitioner, the present paper assists the enterprise modeling effort when the concern is focused on modifiability and enterprise software change. If the metamodel is employed in the beginning of change projects the modeler will get cost estimations and the possibility to highlight risks early.

To the EA tool industry, the present paper hints to potential new features or products. A tool incorporating the provided metamodel would give a qualitative and quantitative support to the user’s modeling effort. The current market for EA tools does not explicitly consider different analyses, especially not with focus on modifiability and change cost.

To the scientific community, the presented metamodel combines the approach of enterprise architecture modeling for analysis with the approaches of modifiability analysis and change cost estimation. These communities can benefit from this work as well as continue contributing in the combined area by: 1) extending/improving the metamodel, 2) conducting more case studies and 3) surveying
more experts.

As discussed in section 7 there are some metamodel elements that after further research perhaps could be reconsidered, i.e. removed or changed. Also, in the surveys and case studies some new elements have been suggested to be included in the metamodel (indicating possible blind spots in existing academic literature). Before any such additions can be made some additional research is needed. Since the current version of the metamodel mainly focuses on change organization, change environment, documentation and software system related issues, special focus could be on the business related elements of enterprise architecture. Many business issues are now implicitly included in the change difficulty attribute of the technical changes class. New classes to add and test might be; management, business organization, requirement specification and business process.

10. Conclusions

Enterprise architecture models can be used to increase the general understanding of enterprise systems and specifically to perform various kinds of analysis. This paper presents a set of instantiated architectural models for enterprise systems modifiability analysis, i.e. for assessing the cost of making changes to enterprise-wide systems. The instantiated architectural models detailed are based on 21 software change projects conducted at four large Nordic companies.

The enterprise architecture models are formalized using probabilistic relational models, combining regular entity-relationship modeling aspects with means to perform enterprise architecture analysis. The presented models contain objects such as change activities, architects and developers, systems and their components, documentation, infrastructure, and a change management process. Each object has a set of attributes related to it. For instance, a component object contains the attributes understandability, coupling, complexity, and size. These attributes are causally related to each other and provide the user with probabilistic analysis capabilities taking uncertainty into consideration.

The paper discusses the validity of the modifiability metamodel based on data collected in workshops and through surveys with both academia and industry (in total 110 experts were surveyed). The studied data shows that the metamodel seems to contain the appropriate elements. By employing the metamodel, data from 21 software change projects was collected. This data was used in order to validate the estimation capabilities of the metamodel, i.e. how well the metamodel estimates software change cost. The metamodel seems to produce estimates within a 75 % accuracy in 87 % of the time and has a mean accuracy of 88 % (when considering projects of 2 000 man-hours or more).
References


37


