Software Qual J (2010) 18:437–468
DOI 10.1007/s11219-010-9100-0

Architecture analysis of enterprise systems modifiability: a metamodel for software change cost estimation

Robert Lagerström · Pontus Johnson · Mathias Ekstedt

Published online: 4 June 2010

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

R. Lagerström (✉) · P. Johnson · M. Ekstedt
Industrial Information and Control Systems, The Royal Institute of Technology,
Osquldas väg 12, 100 44 Stockholm, Sweden
e-mail: robertl@ics.kth.se

P. Johnson
e-mail: pj101@ics.kth.se

M. Ekstedt
e-mail: mek101@ics.kth.se

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 that 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 few standalone applications in the modern software system landscape. Many systems are tightly integrated with others. Furthermore, the management of software systems is no longer an isolated task of one or a 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 enterprises 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 was 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 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 software projects are cancelled before completion, whereas of those completed only 28% were delivered on time, and the average software project overran 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 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 Sect. 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 Sect. 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 Sect. 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 Sect. 8 in order to illustrate the applicability of the presented metamodel. Section 9, discusses research and future work. Finally, Sect. 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 1980s. 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 follows: 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. (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 taxonomy. Oman et al. (1992) 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. (1992) 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 pinpoint 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 international 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 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 following: 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 which brings 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 Sect. 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 into 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, is 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, and for software architecture languages in Allen (1997) where for instance deadlock 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 through 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*, $\mathscr{X} = X_1, \ldots, X_n$. 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 $\mathscr{A}(X)$. 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 $\mathscr{R}(X)$. We use $X.\rho$ to denote the reference slot ρ of X. Each reference slot ρ is typed with the *domain type* $Dom[\rho] = X$ and the *range type* $Range[\rho] = Y$, where $X; Y \in \mathscr{X}$. A slot ρ 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* $\mathscr{I}$ (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* σ_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 Π specifies a probability distribution over all instantiations $\mathscr{I}$ of the metamodel $\mathscr{M}$. 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 $B \in \mathscr{A}(X.\tau)$ and τ is either empty, a single slot ρ or a sequence of slots $\rho_1, \ldots, \rho_k$ such that for all i, $Range[\rho_i] = Dom[\rho_{i+1}]$. For example, the attribute *Cost* of class *Change Project* may have *Change Organization.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, $P(X.A|Pa(X.A))$. For instance, $P(Change\ Project.Cost = high|Change\ Organization.Developer.Expertise = low) = 90\%$ specifies the probability that the project cost will be high, given the expertise of the developers involved.

We can now define a PRM Π for a metamodel $\mathscr{M}$ as follows. For each class $X \in \mathscr{X}$ and each descriptive attribute $A \in \mathscr{A}(X)$, 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 σ_r (i.e. a metamodel instantiated to all but the attribute values), a PRM Π specifies a probability distribution over a set of instantiations $\mathscr{I}$ consistent with σ_r:

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

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

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

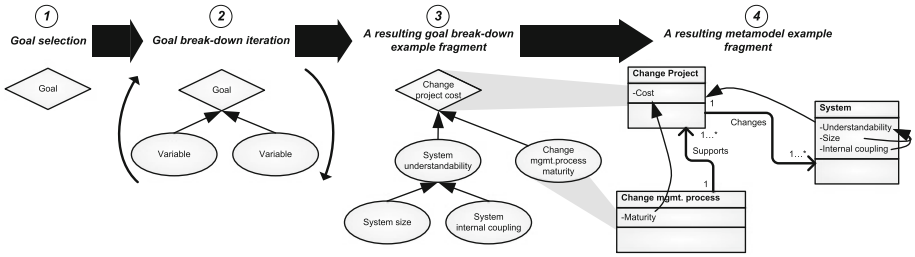

Fig. 1 The iterative goal break-down process, a resulting goal break-down fragment example, and the metamodel fragment related to it

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.

Using the guidelines for metamodel fragment merge, a metamodel was finally obtained for modifiability analysis. Figure 2 depicts the classes and reference slots. Figures 3–6 present different views of the modifiability metamodel including the aggregating functions and classes. Figure 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 preceding 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 to introduce 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*.

$$P(y_j|x_{i,n}) = \begin{cases} \left(z_{1,n} + \frac{2}{3}z_{2,n} + \frac{1}{3}z_{3,n}\right)\frac{1}{z_{1,n}+z_{2,n}+z_{3,n}} & \text{if } i = j \\ \left(\frac{1}{6}z_{2,n} + \frac{1}{3}z_{3,n}\right)\frac{1}{z_{1,n}+z_{2,n}+z_{3,n}} & \text{if } i \neq j \end{cases}$$
$$i \in \{1,2,3\} \ni j$$

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 Sect. 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

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

Comp. change env. tool quality (24)		High				…	…	Low
Comp. change env. infra. quality (25)		High				…	…	Low
Tech. change to comp. change size (23)		Small			Medium	…	…	Large
Components' understandability (22)		Easy	Norm.	Diff.	Easy	…	…	Diff.
Tech. change to comp. change diff.	Easy	0,84	0,61	0,61	0,69	…	…	0,08
	Normal	0,08	0,31	0,08	0,23	…	…	0,08
	Difficult	0,08	0,08	0,31	0,08	…	…	0,84

affecting the *Change difficulty* attribute of the class *Technical changes to components* in Fig. 7.

As described in Sect. 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. Figs. 2–7, 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 cannot 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 Sect. 5.1, the classes and reference slots of the metamodel are presented, and in Sect. 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.

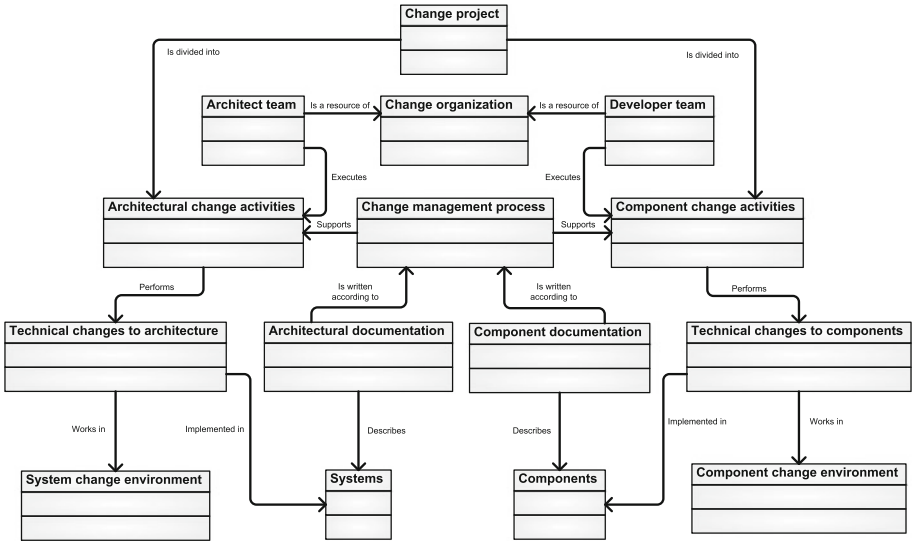

Fig. 2 The modifiability metamodel presenting the main classes and reference slots

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 different components in 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 as follows: (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.

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 Sect. 3 and further explained in Sect. 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 that 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

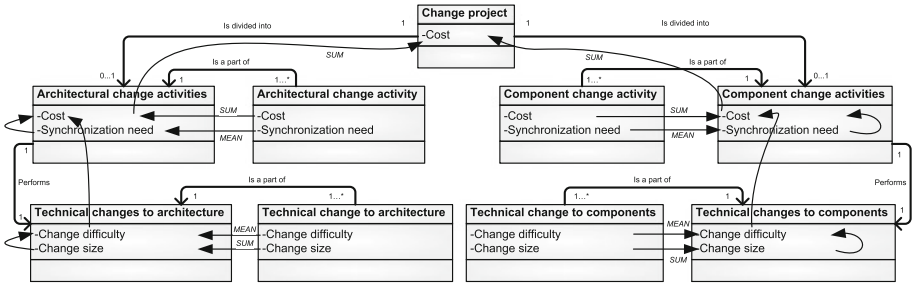

Fig. 3 The project view of the metamodel

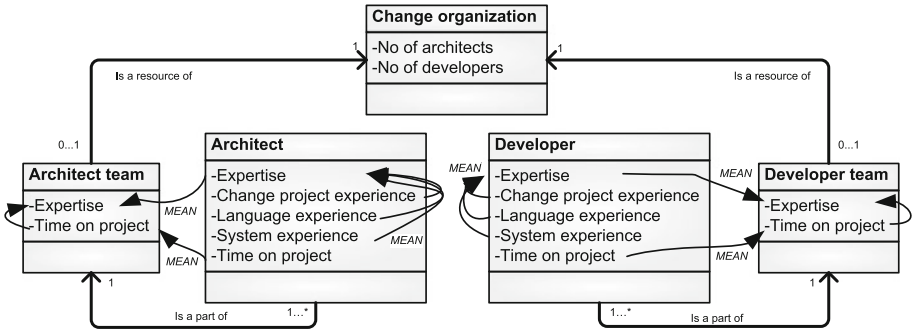

Fig. 4 The organizational view of the metamodel

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 spent on 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

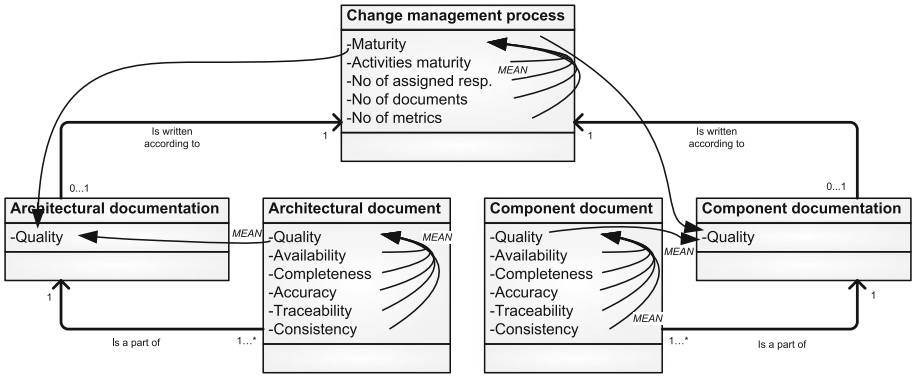

Fig. 5 The documentational view of the metamodel

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 as follows: (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—non existent, 1—initial/ad hoc, 2—repeatable but intuitive, 3—defined process, 4—managed and measurable, and 5—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 {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, $\{0, \ldots, 100\}$.

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

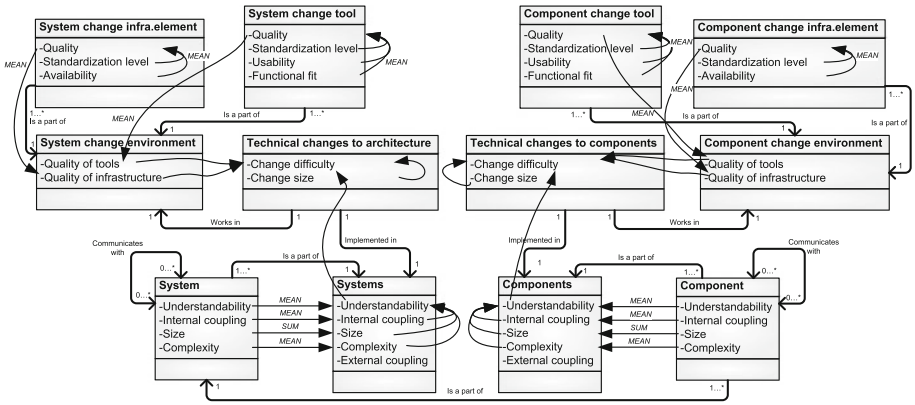

Fig. 6 The system view of the metamodel

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), Matinlassi 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).

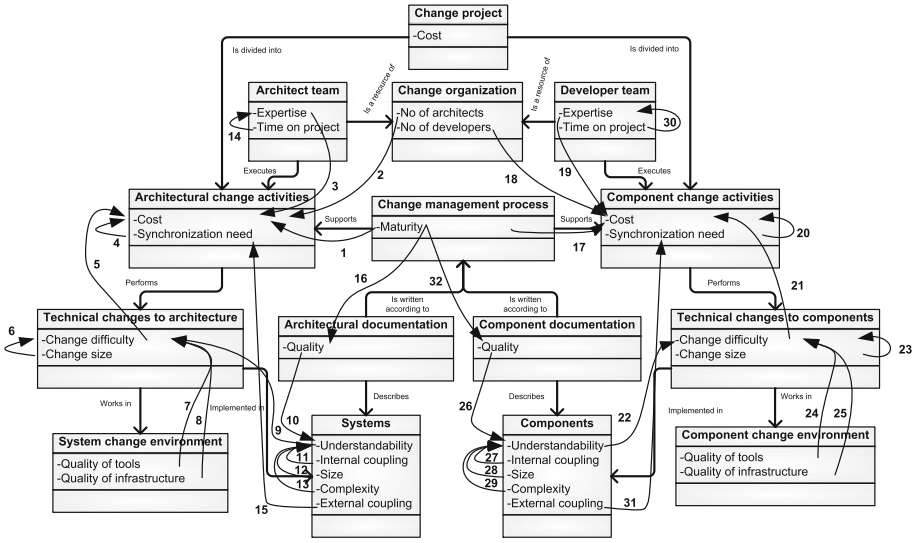

Fig. 7 The modifiability metamodel containing classes and attributes, including numbering of the causal dependencies between attributes

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 Sect. 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. Throughout Sect. 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 Sect. 4.2 and the data collection presented in Sect. 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 a good function in minimizing these uncertainties.

Since the presented metamodel (PRM) is defined based on expert knowledge mapped to a three point 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 as follows: (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 Sects. 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 Sect. 6.3.

Finally, Sect. 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.

Table 2 The workshop and survey results for the question "Which attributes are missing in the metamodel?"

Class	Attribute	No of answers
Management	Degree of support	3
Testing process	Test coverage	2
System	Level of quality goals restrictions	2
	Platform independence	2
Requirements specification	Number of authors	3
	Number of changes during project	2
Architecture goals	Prioritized and communicated	3
Business organization	Stability	2
Change organization	Geographic distance (culture & language diff.)	3
	Understandability of business objects	2
	Use of a common information model	3
Change activities	Tim e restrictions (deadline)	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 agreeing when answering the questionnaire, i.e. they skipped 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.

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. Eleven 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

Table 3 Survey and workshop data regarding the strength of influence between causally related attributes in the metamodel

Relation	High effect	Low effect	No effect	I don't know	No of answers	No eff. percentage (%)
1	25	19	2	6	52	4.5
2	15	26	7	4	52	17.1
3	44	7	0	1	52	0.0
4	22	24	2	4	52	4.3
5	44	7	1	0	52	2.0
6	28	19	0	5	52	0.0
7	29	17	1	5	52	2.2
8	27	18	2	5	52	4.4
9	43	8	1	0	52	2.0
10	37	13	1	1	52	2.0
11	33	14	3	2	52	6.4
12	41	9	1	1	52	2.0
13	44	6	0	2	52	0.0
14	28	20	1	3	52	2.1
15	36	14	1	1	52	2.0
16	27	18	3	4	52	6.7
17	23	13	2	4	42	5.6
18	14	21	1	6	42	2.9
19	38	4	0	0	42	0.0
20	18	8	2	14	42	7.7
21	35	5	0	2	42	0.0
22	36	5	1	0	42	2.4
23	10	25	4	3	42	11.4
24	23	16	1	2	42	2.6
25	23	15	1	3	42	2.6
26	29	12	1	0	42	2.4
27	30	11	1	0	42	2.4
28	27	12	1	2	42	2.6
29	39	2	0	1	42	0.0
30	29	10	1	2	42	2.6
31	31	8	0	3	42	0.0
32	18	15	3	6	42	9.1

The data is also used for defining the CPDs as explained in Sect. 4.2

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. Sect. 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

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

Accuracy is then calculated as $1 - \mathrm{MRE}$. 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

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

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

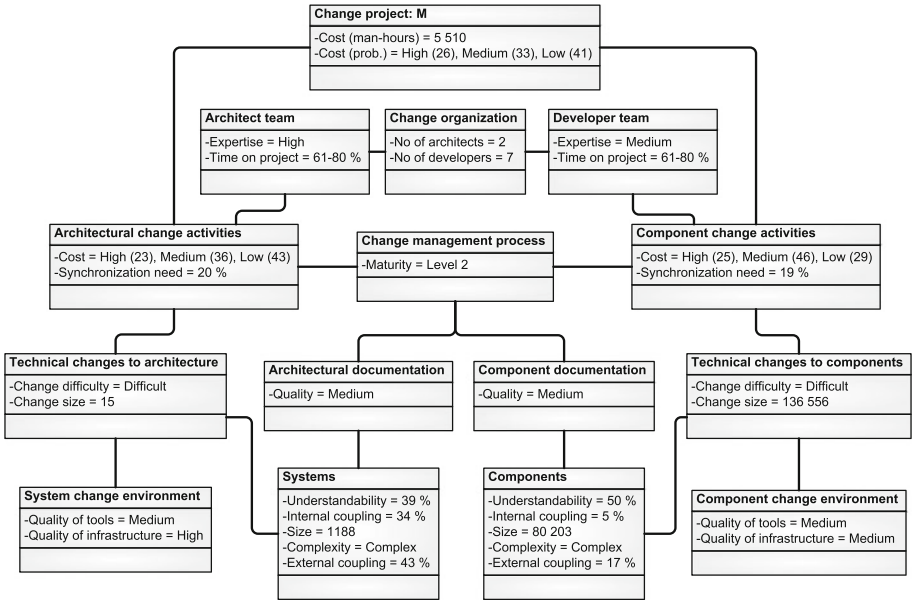

Fig. 8 The main view of the instantiated architectural model containing data for project M of the multiple case study at the Nordic transportation company. The model actually contains many sub-views including the aggregating objects, as the one presented in Fig. 9

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.

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 as follows: 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

$$\mathrm{PRED}(q) = k/n.$$

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

Three of the more well-known and used models and methods are the COnstructive COst MOdel (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.

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 metamodel is 88% (when studying the 15 project above 2,000 man-hours). As we can see, they seem to provide rather similar estimation accuracies.

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.

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

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

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

Model/method	Mean accuracy (%)
The modifiability metamodel, all above 2,000 man-hours	88
Planning poker	82

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 Sect. 2.2 one issue to consider 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.

Focusing on the systems external coupling view of the instantiated architectural model for project M, cf. Fig. 9, we can see that 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

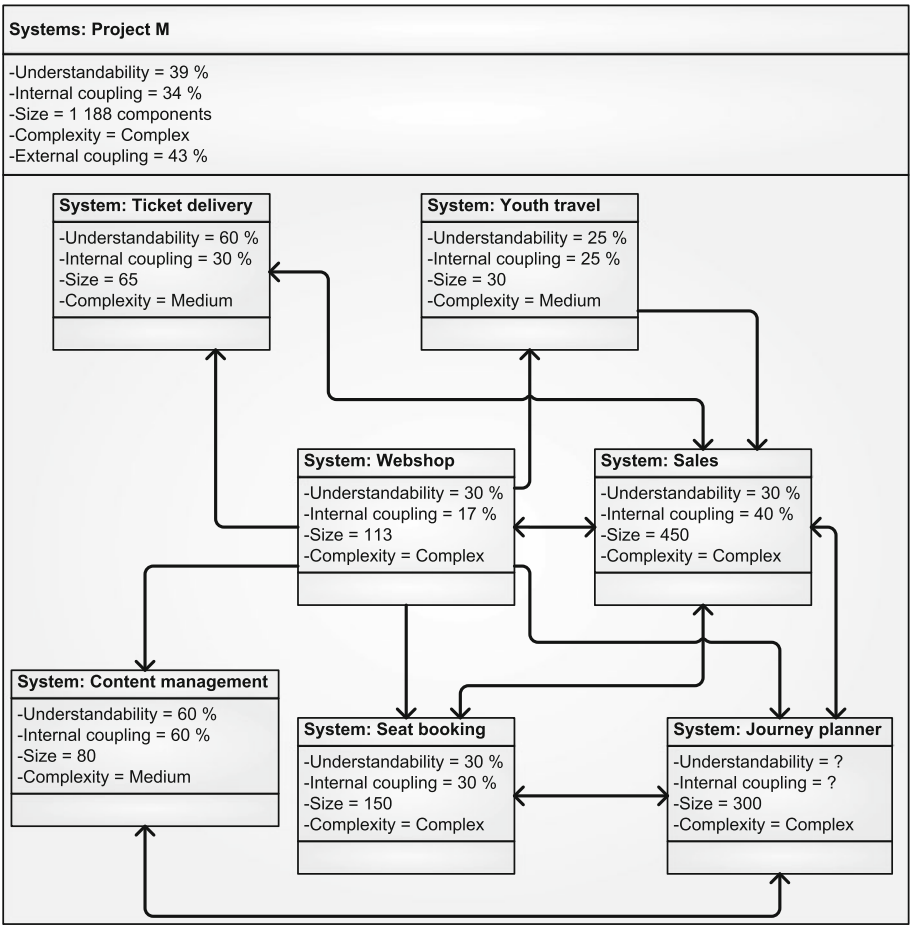

Fig. 9 A systems external coupling view of the instantiated architecture model for project M

journey planing system. In all, there are seven systems, coupled together, involved in the change project, i.e. seven systems in need of modifications.

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

Table 7 Cost intervals for categorization of change project size

Segment\cost	Hgh	Medium	Low
Large	40,000	6,000	1,000
Medium	12,000	6,000	1,000
Small	5,000	2,500	1,000

is done. This has so far only been done a posteriori and further research is being conducted to classify these more objectively and generally.

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 1,188 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. First, 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 analyses. The model is for example able to reveal which parts of the project carry a high risk of cost overruns. 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, they provide a final cost of the projects for comparison, but they also imply an uncertainty of the a priori value of the data. Another issue threatening 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. Besides, there are other methods that in the future could be used to further test and validate the metamodel. An appropriate method could be structural equational 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

Aggarwal, K., Singh, Y., & Chhabra, J. K. (2002). An integrated measure of software maintainability. In *Proceedings of the annual IEEE reliability and maintainability symposium.*

Aier, S., Buckl, S., Franke, U., Gleichauf, B., Johnson, P., Närman, P., Schweda, C., & Ullberg, J. (2009). A survival analysis of application life spans based on enterprise architecture models, enterprise modelling and information systems architectures. In *Proceedings of the 3rd international workshop EMISA*.

Allen, R. (1997). *A formal approach to software architecture*. PhD Thesis, Carnegie Mellon University.

April, A., & Abran, A. (2008). *Software maintenance management*. Hoboken, New Jersey: IEEE Computer Society/John Wiley & Sons.

Bass, L., Clements, P., & Kazman, R. (1998). *Software architecture in practice*. Reading, MA: Addison Wesley Longman/Software Engineering Institute.

Bengtsson, P. O. (2002). *Architecture-level modifiability analysis*. PhD Thesis, Blekinge Institute of Technology.

Boehm, B. (1981). *Software engineering economics*. Upper Saddle River: Prentice Hall.

Boehm, B., Abts, C., & Chulani, S. (2000). Software development cost estimation approaches—a survey. *Annals of Software Engineering, 10*, 177–205.

Brinkkemper, S. (1996). Method engineering: Engineering of information systems development methods and tools. *Information and Software Technology, 38*, 275–280.

Brinkkemper, S. (2000). Method-engineering with web-enabled methods. In *Informations Systems engineering—State of the art and research themes* (pp. 123–133). Berlin: Springer.

Chan, T., Chung, S. L., & Ho, T. H. (1996). An economic model to estimate software rewriting and replacement times. *IEEE Transactions on Software Engineering, 22*.

Chulani, S., Boehm, B., & Steece, B. (1999). Calibrating software cost models using bayesian analysis. *IEEE Transactions on Software Engineering*, 573–583.

Conte, S., Dunsmore, H., & Chen, V. (1985). *Software effort estimation and productivity*. New York: Academic Press Inc.

Department of Defense Architecture Framework Working Group. (2007). DoD architecture framework, version 1.5. Technical Report, Department of Defense, USA.

Fenton, N. E., & Pfleger, S. L. (1997). *Software metrics: A rigorous and practical approach*. Boston, MA: PWS Publishing Company.

Fischer, R., Aier, S., & Winter, R. (2007). Enterprise modelling and information systems architectures. *A Federated Approach to Enterprise Architecture Model Maintenance, 2*, 14–22.

Franke, U., Flores, W. R., & Johnson, P. (2009a). Enterprise architecture dependency analysis using fault trees and bayesian networks. In *Proceedings of 42nd annual simulation symposium (ANSS)* (pp. 209–216), http://www.scs.org.

Franke, U., Höök, D., König, J., Lagerström, R., Närman, P., Ullberg, J., Gustafsson, P., & Ekstedt, M. (2009b). EAF2—a framework for categorizing enterprise architecture frameworks. In *Proceedings of 10th ACIS international conference on software engineering, Artificial intelligence, Networking and Parallel/Distributed Computing* (pp. 327–332).

Friedman, N., Getoor, L., Koller, D., & Pfeffer, A. (1999). Learning probabilistic relational models. In *Proceedings of the 16th international joint conference on artificial intelligence* (pp. 1300–1309).

Granja-Alvarez, J. C., & Barranco-Garcia, M. J. (1997). A method for estimating maintenance cost in a software project: A case study. *Software Maintenance: Research and Practice, 9*, 161–175.

Grubb, P., & Takang, A. (2003). *Software maintenance: Concepts and practice*. Singapore: World Scientific.

Gustafsson, P., Höök, D., Ericsson, E., & Lilliesköld, J. (2009). Analyzing it impacts on organizational structure. In *Portland international center for management of engineering and technology Conference Proceedings*.

Halstead, M. H. (1975). Toward a theoretical basis for estimating programming effort. In *ACM 75: Proceedings of the 1975 annual conference* (pp. 222–224). New York, NY, USA: ACM.

Harrison, W., & Cook, C. (1990). Insights on improving the maintenance process through software measurement. In *Proceedings of the IEEE software maintenance Conference*.

Höök, D., Nordström, L., & Johnson, P. (2009). An enterprise architecture based method for quantified analysis of ict system impact on maintenance management. In *ICOMS asset management conference proceedings*.

IDS Scheer. (2009). Aris business performance edition. Available on http://www.ids-scheer.com/en/ARIS, Accessed 18 June 2009.

IEEE Standards Board. (1990). IEEE standard glossary of software engineering technology. Technical Report, The Institute of Electrical and Electronics Engineers.

International Organization for Standardization. (2001). Software engineering–product quality–part 1: Quality model. International standard ISO/IEC TR 9126–1:2001(E), International Organization for Standardization.

International Organization for Standardization. (2003a). Software engineering–product quality–part 2: External metrics. International standard ISO/IEC TR 9126–2:2003(E), International Organization for Standardization.

International Organization for Standardization. (2003b). Software engineering–product quality–part 3: Internal metrics. International standard ISO/IEC TR 9126–3:2003(E), International Organization for Standardization.

Jarzabek, S. (2007). *Effective software maintenance and evolution: A reuse-based approach*. Boca Raton, FL: Auerbach Publications, Taylor & Francis Group.

Jensen, F. (2001). *Bayesian networks and decision graphs*. New York, Secaucus, NJ, USA: Springer.

Johansson, E. (2005). *Assessment of enterprise information security—How to make it credible and efficient*. PhD Thesis, Royal Institute of Technology (KTH), Stockholm, Sweden.

Johnson, P., & Ekstedt, M. (2007). *Enterprise architecture—Models and analyses for information systems decision making*. Studentlitteratur.

Johnson, P., Lagerström, R., Närman, P., & Simonsson, M. (2007). Enterprise architecture analysis with extended influence diagrams. *Information Systems Frontiers, 9*.

Jürjens, J. (2005). *Secure systems development with UML*. Berlin, Heidelberg: Springer.

Kan, S. (2003). *Metrics and models in software quality engineering* (2nd ed.). London: Pearson Education.

Kazman, R., Asundi, J., & Klein, M. (2001). Quantifying the costs and benefits of architectural decisions. In *Proceedings of the 23rd international conference on software engineering* (pp. 297–306).

Kazman, R., Klein, M., & Clements, P. (2000). ATAM: Method for architecture evaluation.

König, J., & Nordström, L. (2009). Assessing impact of ict system quality on operation of active distribution grids. In *IEEE PowerTech*.

Kurpjuweit, S., & Winter, R. (2007). Viewpoint-based meta model engineering. In *Enterprise modelling and information systems architectures (EMISA 2007)*.

Lagerström, R. (2007). Analyzing system maintainability using enterprise architecture models. *Journal of Enterprise Architecture, 3*, 33–41.

Lagerström, R., Chenine, M., Johnson, P., & Franke, U. (2008). Probabilistic metamodel merging. In *Proceedings of the Forum at the 20th international conference on advanced information systems* (Vol. 344, pp. 25–28).

Lagerström, R., Franke, U., Johnson, P., & Ullberg, J. (2009a). A method for creating enterprise architecture metamodels—applied to systems modifiability analysis. *The International Journal of Computer Science and Applications, 6*, 89–120.

Lagerström, R., Johnson, P., & Höök, D. (2009b). Architecture analysis of enterprise systems modifiability—models, analysis and validation. *submitted to journal*.

Lagerström, R., Johnson, P., Höök, D., & König, J. (2009c). Software change project cost estimation—a bayesian network and a method for expert elicitation. In *the International Workshop on Software Quality and Maintainability Proceeding*.

Lagerström, R., Johnson, P., & Närman, P. (2007). Extended influence diagram generation. In *Interoperability for enterprise software and applications conference*.

Lagerström, R., Saat, J., Franke, U., Aier, S., & Ekstedt, M. (2009d). Enterprise meta modeling methods—combining a stakeholder-oriented and a causality-based approach. In *Enterprise, business-process and information systems modeling*, Lecture Notes in Business Information Processing (Vol 29, pp. 381–393), Berlin Heidelberg: Springer. ISSN 1865-1348.

Laird, L., & Brennan, C. (2006). *Software measurement and estimation: A practical approach*. Hoboken, New Jersey: IEEE Computer Society/John Wiley & Sons.

Lankhorst, M. (2005). *Enterprise architecture at work*. Heidelberg: Springer.

Matinlassi, M., & Niemelä, E. (2003). The impact of maintainability on component-based software systems. In *Proceedings of the 29th IEEE EUROMICRO conference "New Waves in System Architecture"*.

Matson, J., Barrett, B., & Mellichamp, J. (1994). Software development cost estimation using function points. *Software Engineering, IEEE Transactions on, 20*, 275–287.
Moløkken-Østvold, K., Haugen, N. C., & Benestad, H. C. (2008). Using planning poker for combining expert estimates in software projects. *Journal of Systems and Software, 81*, 2106–2117. New York, NY, USA: Elsevier Science Inc.
Närman, P., Johnson, P., & Nordström, L. (2007). Enterprise architecture: A framework supporting system quality analysis. In *Proceedings of the international annual enterprise distributed object computing Conference*.
Neapolitan, R. (2003). *Learning bayesian networks*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.
Oman, P., Hagemeister, J., & Ash, D. (1992). A definition and taxonomy for software maintainability. Technical Report, Software Engineering Lab.
Pigoski, T. (1997). *Practical software maintenance*. New York: John Wiley & Sons.
Putnam, L., & Myers, W. (2003). *Five core metrics*. New York: Dorset House Publishing.
Ross, J. W., Weill, P., & Robertson, D. (2006). *Enterprise architecture as strategy: Creating a foundation for business execution*. Boston, MA: Harvard Business School Press.
Simonsson, M. (2008). *Predicting IT governance performance: A method for model-based decision making*. PhD Thesis, Royal Institute of Technology (KTH), Stockholm, Sweden.
Smith, D. (1999). *Designing maintainable software*. Berlin: Springer.
Society of Automotive Engineers. (2009). Architecture analysis and design language (aadl) standard. Technical Report, Carnegie Mellon University.
Tekinerdogan, B. (2004). ASAAM: aspectual software architecture analysis method. In *Proceedings of the fourth working IEEE/IFIP conference on software architecture* (pp. 5–14). IEEE Computer Society.
Telelogic-IBM. (2009). Ibm rational system architect. Available on http://www.telelogic.com/Products/systemarchitect/, Accessed 15 June 2009.
The IT Governance Institute. (2007). Control objectives for information and related technology (COBIT) 4.1. Technical Report, The IT Governance Institute.
The Open Group. (2009). *The Open Group Architecture Framework (TOGAF)—version 9*. The Open Group.
Troux Technologies. (2009). Troux 8. Available on http://www.troux.com, Accessed June 15, 2009.
Ullberg, J., Lagerström, R., & Johnson, P. (2008). A framework for service interoperability analysis using enterprise architecture models. *IEEE International Conference on Services Computing*.
Warner, R. (2008). *Applied statistics—From bivariate through multivariate techniques*. Beverley Hills, CA: Sage Publications Inc.
Winter, R., & Fischer, R. (2007). Essential layers, artifacts, and dependencies of enterprise architecture. *Journal of Enterprise Architecture, 3*, 7–18.
Zachman, J. (2009). The Zachman framework—the official concise definition. Available on http://www.zachmaninternational.com, Accessed 16 June 2009.
Zachman, J. A. (1987). A framework for information systems architecture. *IBM Systems Journal, 26*, 276–292. Riverton, NJ, USA.
Zuse, H. (1997). *A framework of software measurement*. Berlin: Walter de Gruyter.

Author Biographies

Robert Lagerström received his Msc degree in computer science from the Royal Institute of Technology in late 2005. In early 2006, he started his PhD at the department Industrial Information and Control Systems at the Royal Institute of Technology in Stockholm, Sweden. His topic of research as a PhD-student is Enterprise Architecture and Systems Modifiability. Robert is responsible for the courses "Industrial Information Systems, System Technology" and "Industrial Information Systems, Case Studies". In addition to that he supervises master thesis students. Robert was a member of the Swedish Chapter committee of INCOSE (International Council on Systems Engineering) up until 2009. He is responsible for the Software Metrics network at the Swedish Computer Society. Robert has written a number of academic publications in the field of Enterprise Architecture and Modifiability, also he is a co-author of the book Enterprise Architecture: Models and Analyses for Information Systems Decision Making. Robert is partner and consultant at Management Doctors, a Swedish IT-management consultancy firm.

Pontus Johnson is Professor and Head of the Department of Industrial Information and Control Systems at the Royal Institute of Technology (KTH) in Stockholm, Sweden. Active at the department are 25 researchers and PhD students focusing particularly on the analysis of architectural models of information systems and their context. He is secretary of the IFIP Working Group 5.8 on Enterprise Interoperability, technical coordinator of the FP7 Viking project, organizer, program committee member, and associate editor of several international conferences, workshops, and journals. Pontus supervises a number of PhD students. In his research, he has much contact with Swedish corporations and organizations in the form of research projects, master thesis projects, seminars, and consultations. He has written a book with the title Enterprise Architecture: Models and Analyses for Information Systems Decision Making, available in many book stores. He received his MSc from the Lund Institute of Technology in 1997 and his PhD and Docent title from the Royal Institute of Technology in 2002 and 2007. He was appointed professor in 2009.

Mathias Ekstedt received his MSc in electrical engineering and PhD in Industrial Information and Control Systems from the Royal Institute of Technology in Stockholm, Sweden in 1999 and 2004, respectively. He is currently a research associate at the Royal Institute of Technology and the manager of the program IT Applications in Power System Operation and Control within the Swedish Centre of Excellence in Electric Power Engineering. His research interests include information and control systems and enterprise architecture for the power industry. Mathias is a member of the Task force on Cyber Security of Power Systems within IEEE PES Technical Committee on Power System Analysis, Computing and Economics Subcommittee of Computer and Analytical Methods. He is a member of INCOSE and a former member of the Swedish INCOSE Chapter board. He is also the founder of the enterprise architecture network at the Swedish Computer Society.