Assessing Modifiability in Application Services Using Enterprise Architecture Models – A Case Study

Magnus Österlind, Robert Lagerström, and Peter Rosell

Industrial Information and Control Systems, KTH – The Royal Institute of Technology, Osquldas väg 10, 10044 Stockholm, Sweden
{magnuso,robertl}@ics.kth.se

Abstract. Enterprise architecture has become an established discipline for business and IT management. 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. 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. However, few initiatives explicitly state how one can analyze the EA models in order to aid decision-making. One approach that does focus on analysis is the Enterprise Architecture Modifiability Analysis Tool. This paper suggests changes to this tool and presents a case study in which these have been tested. The results indicate that the changes improved the tool. Also, based on the outcome of the case study further improvement possibilities are suggested.

Keywords: Enterprise architecture, Modifiability analysis, Modeling, Decision-making.

1 Introduction

Enterprise Architecture (EA) has become an established discipline for business and IT management [1] with many initiatives. A few of these initiatives are The Department of Defense Architecture Framework (DoDAF)[2], The Zachman framework [3], The Open Group Architecture Framework (TOGAF) [4] and ArchiMate [5]. EA describes the fundamental artifacts of business and IT as well as their interrelationships [1, 3, 4, 5]. 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. A main concept in EA is the metamodel which acts as a pattern for the instantiation of the architectural models. In other words, a metamodel is a description language used when creating models [4, 5]. 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 [6].

A changing business environment requires IT systems which can be adapted to new conditions. Also, systems in a modern enterprise are often connected to

S. Aier et al. (Eds.): TEAR 2012 and PRET 2012, LNBIP 131, pp. 162–181, 2012.

other systems. This, along with many other factors make it difficult to estimate the effort of modifying a system. Thus, there is a need for decision makers to be able to analyze how much effort is required to modify an enterprise IT system. Such an effort estimation could help the decision maker to assign the resources in a more efficient manner. Combining EA and modifiability analysis has the advantages of being able to analyze the IT systems in their enterprise wide context. [7] proposes The Enterprise Architecture Modifiability Analysis Tool (TEAMATe) as one way of combining EA modeling with formal analysis to solve the difficulties of estimating IT change costs in an enterprise–wide context.

TEAMATe has been tested and validated in four multiple case studies [8]. However, during these case studies it became evident that there are also numerous parts of TEAMATe that need improvements. The contribution of this paper is to present and test some of these improvements. Firstly, attributes related to modifiability analysis have been revised and further developed. Secondly, the formal analysis language used has been revised and changed. Thirdly, classes and class relationships related to modifiability analysis have been aligned with the ArchiMate modeling language . Finally, software tool support has been developed. These improvements have been employed in a case study conducted at KTH – the Royal Institute of Technology investigating nine of the software systems used by KTH University Administration division for Student Records and Information Systems. This case study indicates that the improvements made to TEAMATe was a step in the right direction towards a more usable tool for modifiability analysis using architectural modeling. The study also provided further insights for future development of the tool.

The remainder of the paper is structured as follows: In section 2 related work is presented. Section 3 presents the Enterprise Architecture Modifiability Analysis Tool. The following section goes through the improvements of this tool. Next, in section 5 the case study and its results are described. Section 6, discusses the improvements and results. Finally, section 7 summarizes the paper with conclusions and future work.

2 Related Work

2.1 Enterprise Architecture for Modifiability Analysis

Enterprise architecture has grown into a modeling discipline widely recognized with many initiatives. However, 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 straight forward manner. The Department of Defense Architecture Framework (DoDAF) [2] 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 [3], does connect model types describing different aspects (Data, Function, Network, People, Time, and Motivation) with very abstractly described stakeholders (Strategists, Executive Leaders, Architects, Engineers, and Technicians), but does not provide any deeper insight how different models should be used. The Open Group Architecture Framework (TOGAF) [4], explicitly states stakeholders and concerns for each viewpoint they are suggesting. However, neither the exact metamodel nor the mechanism for analyzing the stated concerns, are described. In relation to modifiability, the most appropriate viewpoints provided would, according to TOGAF, arguably be *the Software Engineering View*, *the Systems Engineering View*, *the Communications Engineering View*, and *the Enterprise Manageability View*. In the descriptions of these views one can find statements such as; "the use of standard and self-describing languages, e.g. XML, are good in order to achieve easy to maintain interface descriptions". However, the exact interpretation of such statements when it comes to architectural models or how it relates to the modifiability of a system as a whole, is left out. Moreover, these kinds of "micro theories" are only exemplary and do not claim to provide a complete theory for modifiability or similar concerns.

2.2 Modifiability Analysis Methods

The issue of dealing with modifiability is not an enterprise architecture specific problem. Managing and assessing IT 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), and the Oman taxonomy.

COCOMO, COnstructive COst MOdel, was in its first version released in the early 1980's. It became one of the most frequently used and most appreciated IT 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. Effort estimation with COCOMO is based on the size of the software, an approximate productivity constant A, an aggregation of five scale factors E (precentedness, development flexibility, architecture/risk resolution, team cohesion, and process maturity), and effort multipliers to 15 cost driving attributes [9].

Bass et al. propose the Software Architecture Analysis Method (SAAM) for software quality evaluation [10]. This method takes several quality attributes into consideration; performance, security, availability, functionality, usability, portability, reusability, testability, integrability and modifiability. Bass et al. categorize modifications as: extending or changing capabilities, deleting unwanted capabilities, adapting to new operating environments and restructuring. Based on the quality attributes presented, Bass et al. propose different architectural styles which then are employed in the SAAM. It is a scenario-based approach which intends to make sure that stakeholder quality goals are met (for instance high

modifiability). According to [10] 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 system.

The Definition and Taxonomy for Software Maintainability presented in [11] provides a hierarchical definition of software maintainability in the form of a taxonomy. [11] found three broad categories of factors influencing the maintainability of an IT system; management, operational environment, and the target system. Each of these top-level categories is then further broken down into measurable attributes. According to [11] the taxonomy can be useful for developers by defining characteristics affecting the software maintenance cost of the software they are developing. Hence, the developers can write highly maintainable software from the beginning by studying the taxonomy. Maintenance personnel can use the taxonomy to evaluate the maintainability of the software they are working with in order to pin point risks etc. Project managers and architects can use the taxonomy in order to prioritize projects and locate areas in need of re-design.

2.3 Related Work Summary

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

Approaching the same issue but from the other direction, the EA frameworks available do not provide any formal analysis mechanisms (especially none for modifiability analysis).

3 The Enterprise Architecture Modifiability Analysis Tool

This section presents the core of TEAMATe, namely the metamodel. The metamodel is a so called PRM (Probabilistic Relational Model), i.e. a metamodel where the attributes are related to each other by causality. These causal relations are defined as conditional probabilities. More information about the metamodel, incl. the attributes, and the PRM formalism can be found in [7]. More information on how to employ the metamodel in specific modeling scenarios, the analysis, and the results can be found in [8].

TEAMATe's metamodel for modifiability analysis, cf. Figure 1, focuses on the IT systems and the surrounding environment involved in or affected by the

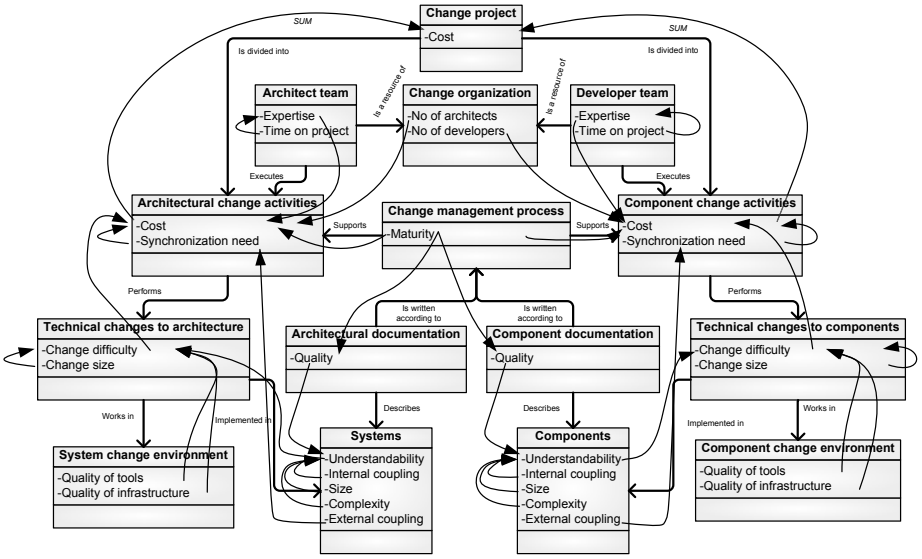

Fig. 1. The TEAMATe metamodel

modifications implemented in a change project, thus aiming at analyzing modifiability defined as change project cost (high modifiability leads to low change costs).

TEAMATe can be divided into three viewpoints: The *architectural viewpoint*. This viewpoint focuses on modeling the as-is and to-be architectures. The two classes System and Component in the metamodel belong to this viewpoint. The *change management viewpoint*. This viewpoint focuses on the change management process and organization. The metamodel contains the classes Change management process, Documentation, and Change Organization which belongs to this viewpoint. The *change project viewpoint*. This viewpoint focuses on the specific change project being analyzed, e.g. analyzing the gap between the as-is architecture and a to-be architecture using the resources of the change management process and organization. That is, the change project viewpoint uses an instantiated subset of the architectural and change management viewpoints. The classes Change project, Team, Change activities, Technical changes, and Change environment belong to the change project viewpoint.

The focus of this paper is to present improvements to the architectural viewpoint of TEAMATe. That is, the emphasis is on the classes related to this viewpoint, System and Component, and the attributes related to these two classes, Understandability, Size, Complexity, and Coupling.

Understandability of a system or a component is measured as the percentage of time spent on trying to understand the system or component in question (in relation to the total time spent on each system/component), $\{0, \ldots, 100\}$. Component size is measured in number of lines of code, and system size is defined

as number of components, $\{0, \ldots, \infty\}$. Complexity is measured subjectively as {Complex, Medium, Not complex}. The system external coupling attribute is defined as the number of actual relations between the systems divided by the number of possible relations between the systems, $\{0, \ldots, 100\}$. System internal and component external coupling are measured as the number of actual relations between the components in the system divided by the number of possible relations between the components, $\{0, \ldots, 100\}$. Component internal coupling is defined as the number of actual relations within the component divided with the number of possible relations, $\{0, \ldots, 100\}$.

3.1 Previous Case Study Findings

TEAMATe has been tested in four multiple case studies (21 change projects) [8]. These cases have provided valuable input for TEAMATe in terms of showing that the approach is useful and worth further development. Some of the needed improvements found during these case studies were:

1) In the architectural viewpoint, TEAMATe contains the attributes coupling, complexity, size, and understandability. However, it was found that these needed to be reviewed in a second iteration. Complexity is only measured subjectively based on peoples experience, although there are objective complexity metrics that can be used. Size is measured as lines of code without taking the chosen programming language into consideration. The coupling measure is not based on the most well-known and tested metrics available. Understandability is defined as an a posteriori measure, which obviously limits its impact on the prediction abilities of TEAMATe. Thus, there is a need of improving the attributes in the architectural viewpoint. See section 4.1.

2) The formal analysis language used, Probabilistic Relational Models (PRMs), is based on probabilities and use Bayesian statistics to calculate estimated values for attributes in the instantiated models. This has in many cases proven useful, however in the architectural viewpoint most attributes are not related by causality and the values can be calculated based on information already available in the model. E.g. the size of a system can be calculated based on the size of its components. Thus, the architectural viewpoint would benefit from having the analysis language revised. See section 4.2.

3) ArchiMate is a commonly used modeling language and many companies can therefore relate to its concepts. Thus alignment with ArchiMate would be beneficial. See section 4.3.

4) In the multiple case studies three different tools were employed. The data collected was stored with Microsoft Excel. The architecture modeling was done with Microsoft Visio. The probabilistic analysis was carried through with GeNIe. This manual integration of tools ended up being a heavy workload for the modeler. TEAMATe would thus be more user-friendly if there was a modeling tool that could handle collected data, modeling, and analysis. See section 4.4.

4 TEAMATe: The Next Generation

This section presents the new and improved version of the architectural viewpoint in the Enterprise Architecture Modifiability Analysis Tool.

4.1 Metrics

Complexity. IEEE defines complexity as *the degree to which a system or component has a design or implementation that is difficult to understand and verify* [12]. Halstead's complexity metric was introduced in 1977 [13], it is based on the number of operators (e.g. and, or, while) and operands (e.g. variables and constants) in a software program. A drawback of Halstead's complexity metric is that it lacks predicting power for development effort since the value can be calculated first after the implementation is complete [14]. Information flow complexity, IFC, as presented in [15] is based on the idea that a large amount of information flows is caused by low cohesion, low cohesion is in turn causing a high complexity. One problem with the IFC metric is that it produces a lower complexity value for program code using global variables compared to a solution which uses function arguments when called, this is contradicting to common software design principles [16]. In this paper McCabe's Cyclomatic Complexity (MCC) metric is employed [17]. [14] has identified that MCC is useful to, identify overly complex parts of code, identify non-complex part of code, and to estimate maintenance effort. MCC is based on the control structure of the software, the control structure can be expressed as a control graph. The cyclomatic complexity value of a system with the control graph G is calculated with the following equation: $v(G) = e - n + 2$ or equivalently $v(G) = DE + 1$ where e=number of edges in the control graph, n=number of nodes in the control graph, DE=number of predicates. Considering the example code presented in Figure 2 the control graph G_{sort} can be obtained.

The MCC value of G_{sort} (cf. Figure 2) is $v(G_{sort}) = 14 - 12 + 2 = 4$. McCabe has performed a study indicating that the cyclomatic complexity value of a component should be kept below 10 [17]. MCC has been used in other studies providing additional complexity levels and guidelines on how complex a piece of software code is [14]:

- 1-4, a simple procedure.
- 5-10, a well-structured and stable procedure.
- 11-20, a more complex procedure.
- 21-50, a complex procedure, worrisome.
- 50<, an error-prone, extremely troublesome, untestable procedure.

Size. Lines of code (LOC) and function points (FP) are two ways to measure the size of an IT system. FP are based on the inputs, outputs, interfaces and databases in a system [14]. FP have the advantage of being technology independent, reasonably reliable and accurate, and they are effective from an early stage of the IT system life cycle [14]. The disadvantages of the FP size metric is

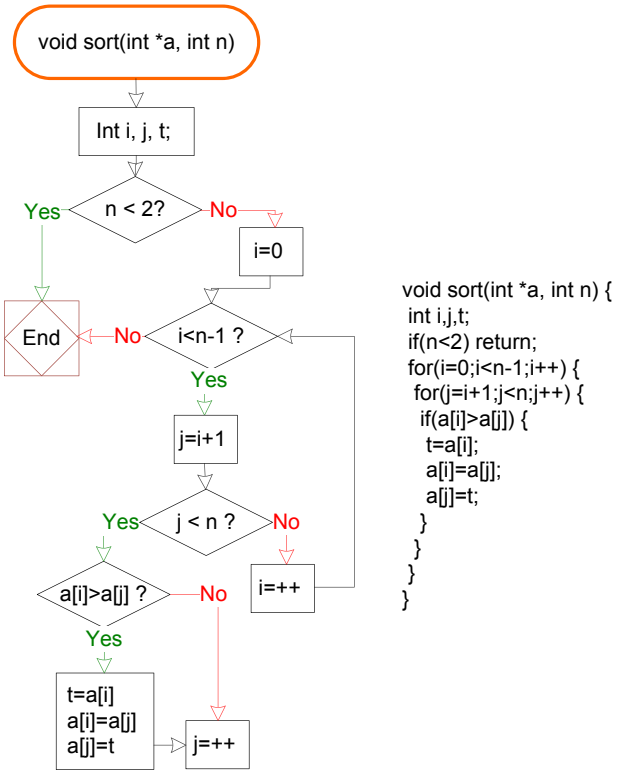

Fig. 2. An example of a control graph G_{sort} with $v(G_{sort}) = 4$

that it requires significant effort to derive [14]. The LOC in a system provides the core functionality and can therefore be of importance when estimating how easy it would be to implement changes to the system. LOC in a system can be measured in different ways: Using source lines of code (SLOC) every line of code in the software implementation is counted. Non-commented lines of code (NLOC) is a subset of the previous option, where the blank lines and comments are excluded. Logical lines of code (LLOC) is another approach, where only the executable statements of the software are counted. The most popular option is NLOC, however the most important thing is to be consistent with the way you measure [14]. No matter which LOC measure is used it needs to be well specified to provide a reliable measurement [14]. A framework on how to measure lines of code has been created by the Software Engineering Institute of Carnegie-Mello University [18], with the aid of this framework the LOC measure can be specified to provide a coherent way of how to measure LOC.

Aivosto[1] suggests a classification of system size for systems coded with Visual Basic 1. The classification is based on long-time experience, but has not been

[1] `http://www.aivosto.com/project/help/pm-loc.html`

validated making it less reliable. However, given the size of the systems studied in [19], the classification seems trustworthy. Related to system size, operating systems can be much larger with over 40 million LOC [20], however an operating system would not be modeled as an application that an enterprise wishes to modify. Rumor has it that SAP[2] has over 250 million LOC in their product portfolio [3], but we believe that no enterprise would model SAP as one application. Thus, the classification of system size by Aivosto[1] still seems appropriate for our purpose.

Table 1. System size classification

Classuification	LOC
Small	0-9.999
Medium	10.000-49.999
Semi-large	50.000-99.999
Large	100.000-499.999
Very large	500.000≤

Since different programming languages are more or less expressive per line of code [14], a gearing factor can be used when comparing the lines of code of two systems if they are created in different programming languages. A high gearing factor value indicates poor expressiveness, hence a programming language with a low gearing factor require less lines of code to implement a function; given that the language is appropriate to use. [21] has published gearing factors for some programming languages of which a subset is presented in Table 2.

Table 2. Gearing factors for some commonly used programming languages

Language	Gearing factor
Assembly-Basic	320
C#	59
C++	55
Java	53
Visual Basic	52
ASP	50

Coupling. IEEE has defined coupling as *the manner and degree of interdependence between software modules. Types include common-environment coupling, content coupling, control coupling, data coupling, hybrid coupling, and pathological coupling* [12]. Fenton and Melton have developed a coupling metric based on Myers coupling levels [22], these levels are:

[2] SAP AG, an IT system vendor.
[3] `http://judithbalancingact.com/2007/04/30/`

- Content coupling relation $R_5 : (x, y) \in R_5$ if x refers to the internals of y , i.e., it branches into, changes data, or alters a statement in y.
- Common coupling relation $R_4 : (x, y) \in R_4$ if x and y refer to the same global variable.
- Control coupling relation $R_3 : (x, y) \in R_3$ if x passes a parameter to y that controls its behavior.
- Stamp coupling relation $R_2 : (x, y) \in R_2$ if x passes a variable of a record type as a parameter to y , and y uses only a subset of that record.
- Data coupling relation $R_1 : (x, y) \in R_1$ if x and y communicate by parameters, each one being either a single data item or a homogeneous set of data items that does not incorporate any control element.
- No coupling relation $R_0 : (x, y) \in R_0$ if x and y have no communication, i.e., are totally independent.

The Fenton and Melton coupling measure is pairwise calculated between components,

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

where n= number of interconnections between x and y. i= level of highest (worst) coupling type found between x and y.

Modifiability. To evaluate the modifiability, the complexity levels by [14], the coupling levels by [22] and the size levels by Aivosto[1] are used in order to indicate how "good" a modeled architecture is. The highest level is given a value of 0, meaning that the modifiability is low, and the lowest level a value of 5, a good modifiability. The modifiability level is then evaluated as the sum of the three individual metrics. The reason to summarize the values is to create a metric that can be used in order to indicate whether a system is likely to be easy to modify or not. According to the correlations levels in [16] the three metrics used are more or less equally important when estimating the level of modifiability in an IT system. The modifiability metric gives a rough estimation, which can be of value when making decisions regarding different architecture scenarios.

The attributes in the architectural viewpoint of TEAMATe are revised based on these four metrics, cf. section 4.5 for the new metamodel description.

4.2 Analysis Language

P2AMF. The Object Constraint Language (OCL) is a formal language typically used in order to describe constraints on UML models [23]. Additionally, OCL can be applied for queries over objects described in a model [24].

The Predictive, Probabilistic Architecture Modeling Framework (P2AMF) is a probabilistic Object Constraint Language. P2AMF is an extension of OCL for probabilistic analysis and prediction, first introduced in [25] (under the name Pi-OCL). The main feature of P2AMF is its ability to express uncertainties of objects, relations and attributes in UML-models and perform probabilistic analysis incorporating these uncertainties, as illustrated in [25].

An example usage of P2AMF could be to create a model for calculating the coupling between application components, a P2AMF expression to calculate the coupling with the Fenton and Melton metric [22] might look like this:

context ApplicationComponent: attribute coupling : Real = let n : Integer = numberOfInterconnectionsBetweenPair(y) in getWorstCouplingTypeInPair(y) + n/(n+1)

Since the attributes in the architectural viewpoint are not related by causality, the PRM formalism seems less appropriate to use here. With P2AMF attributes can be evaluated based on information in the model and it is easy to for instance calculate the modifiability level with the definition given in the previous subsection. This would not have been possible with PRMs.

Therefore, the analysis formalism used in the architectural viewpoint of TEAMATe has been revised and is using the P2AMF as analysis language instead of PRMs, cf. section 4.5 for the P2AMF statements defining the metamodel.

4.3 ArchiMate Alignment

ArchiMate. Lankhorst [5] has probably published the most well-known and widespread metamodel, called ArchiMate. The ArchiMate metamodel is an open, independent, and general modeling language for enterprise architecture. The primary focus of ArchiMate is to support stakeholders how to address concerns regarding their business and the supporting IT systems. ArchiMate is extensively presented in [5] and is partly based on the ANSI/IEEE 1471-2000, Recommended Practice for Architecture Description of Software-Intensive Systems, also known as the IEEE 1471 standard [26]. The Open Group accepted the ArchiMate metamodel as a technical standard[4]. The ArchiMate metamodel consists of three layers; the Business layer, the Application layer and the Technology layer. Where the technology supports the applications, which in turn support the business. Each layer consists of a number of classes and defined class relationships. The classes in each layer are categorized into three aspects of enterprise architecture: 1) The passive structure - modeling informational objects. 2) The behavioral structure - modeling the dynamic events of an enterprise. 3) The active structure - modeling the components in the architecture that perform the behavioral aspects. Figure 3 presents the application layer of the ArchiMate metamodel.

Four classes were identified as useful in the application layer, namely; Application Service, Application Function, Application Component, and Application Collaboration. These classes have thus replaced the two previously used classes System and Component. Also three new class relationships where added to the metamodel based on the relationships in ArchiMate, namely; Realization, Assignment, and Collaboration.

ArchiMate [5] defines: An *Application Service* as an externally visible unit of functionality, provided by one or more components, exposed through well-defined interfaces, and meaningful to the environment. An *Application Function* as a behavior element that groups automated behavior that can be performed by an application component. An *Application Component* as a modular, deployable,

[4] `http://www.opengroup.org/archimate`

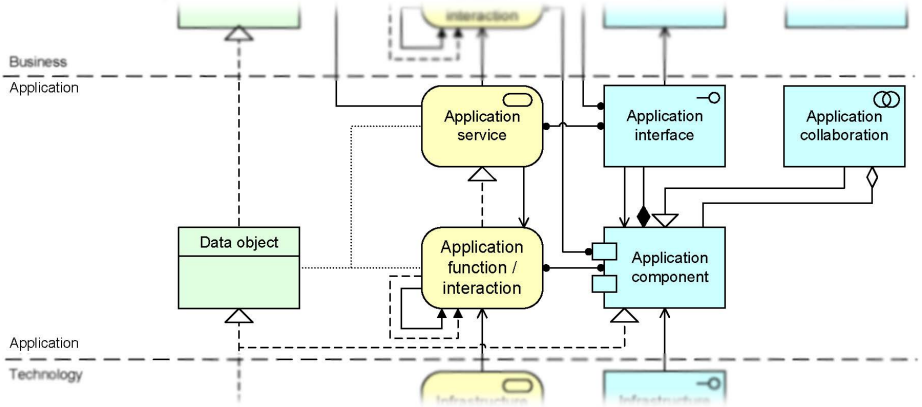

Fig. 3. The application layer of the ArchiMate metamodel

and replaceable part of a system that encapsulates its contents and exposes its functionality through a set of interfaces. An *Application Collaboration* as an aggregate of two or more application components that work together to perform collective behavior.

Realization relationship. The realization relationship links a logical entity with a more concrete entity that realizes it. In the metamodel the realization relationship is used between the application function and application service in order to model what functions realize a service. *Assignment relationship.* The assignment relationship links active elements with units of behavior that are performed by them. In the metamodel the assignment relationship is used to model the performed behavior of an application component that is an application function. *Collaboration relationship.* In the metamodel the collaboration relationship links an application collaboration with two application components. The collaboration relationship is used to model the execution dependencies between two application components or two application components charing information.

With these classes and relationships the architectural viewpoint of TEAMATe is considered to be aligned with ArchiMate, cf. section 4.5 for the metamodel description including the attributes populating these classes.

4.4 Tool Support

EAAT. The Enterprise Architecture Analysis Tool (EAAT[5]), is a software tool which supports enterprise architecture modeling and formal analysis using P2AMF. EAAT supports creation of metamodels with built in P2AMF analysis functionality. The metamodel can then be instantiated into models which are then analyzed based on the analysis code from the metamodel. One advantage of EAAT is that collected data is stored as evidence in the models. EAAT has been used in other case studies such as [25].

[5] `http://www.ics.kth.se/eaat`

4.5 The New Metamodel

This subsection presents the new and improved architectural viewpoint metamodel of TEAMATe, containing the revised metrics using P2AMF and ArchiMate modeled in EAAT.

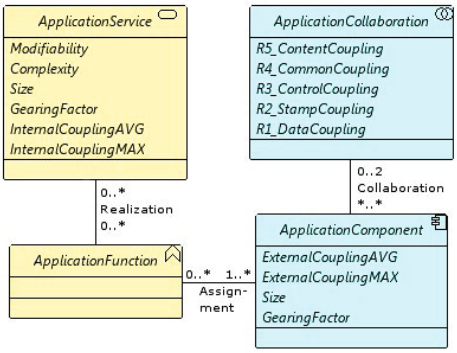

Fig. 4. The new version of the architectural viewpoint metamodel in TEAMATe

Application Service. The application service class contains the attributes Modifiability, Complexity, Size, Gearing Factor, and Coupling.

Attribute: Modifiability The modifiability metric is an aggregation of the attributes: Complexity α, InternalCouplingMAX β, and Size γ.

The complexity levels from [14] are used to give complexity c a numerical value α, where $0 \leq \alpha \leq 5$.

- If ApplicationService.Complexity is $c = 0$, then $\alpha = 5$.
- If ApplicationService.Complexity is $1 \leq c \leq 4$, then $\alpha = 4$.
- If ApplicationService.Complexity is $5 \leq c \leq 10$, then $\alpha = 3$.
- If ApplicationService.Complexity is $11 \leq c \leq 20$, then $\alpha = 2$.
- If ApplicationService.Complexity is $21 \leq c \leq 50$, then $\alpha = 1$.
- If ApplicationService.Complexity is $50 < c$, then $\alpha = 0$.

The coupling levels from [22] are used to give internal coupling (max) icm a numerical value β, where $0 \leq \beta \leq 5$.

- If ApplicationService.InternalCouplingMax is ≤ 1, then $\beta = 5$.
- If ApplicationService.InternalCouplingMax is $1 \leq icm < 2$, then $\beta = 4$.
- If ApplicationService.InternalCouplingMax is $2 \leq icm < 3$, then $\beta = 3$.
- If ApplicationService.InternalCouplingMax is $3 \leq icm < 4$, then $\beta = 2$.
- If ApplicationService.InternalCouplingMax is $4 \leq icm < 5$, then $\beta = 1$.
- If ApplicationService.InternalCouplingMax is $5 \leq icm$, then $\beta = 0$.

The size levels from Aivosto[1] are used to give size s a numerical value γ, where $0 \leq \gamma \leq 5$.

- If ApplicationService.Size is $s < 10.000$, then $\gamma = 4$.
- If ApplicationService.Size is $10.000 \leq s < 50.000$, then $\gamma = 3$.
- If ApplicationService.Size is $50.000 \leq s < 100.000$, then $\gamma = 2$.
- If ApplicationService.Size is $100.000 \leq s < 500.000$, then $\gamma = 1$.
- If ApplicationService.Size is $s \leq 500.000$, then $\gamma = 0$.

If S is an application service, then the modifiability value $= \alpha + \beta + \gamma$ with $\mathbf{V}(S.Modifiability) = \{x \in \mathbb{N} : 0 \leq x \leq 14\}$. A low modifiability value indicates that an application service is difficult to change just as a high modifiability value indicates the opposite.

Attribute: Complexity. The complexity attribute is calculated as the cyclomatic complexity by McCabe [17]. The application components are used as nodes and the values in the attributes of the application collaboration class are used as edges. This includes relations to an application component outside of the owning application service, in the case an application collaboration exists between one application component realizing the service is collaboration with a component realizing a different application service. If $I = \{i_1, ..., i_n\}$ is a list of application collaborations, S is an application service, c is an application component where $c \subseteq S.realize$ and $I \subseteq S.realize.collaboration$, then

$$f(S.Complexity) = \sum_{i=1}^{n} i_i.R5\_ContentCoupling + \sum_{i=1}^{n} i_i.R4\_CommonCoupling$$

$$+ \sum_{i=1}^{n} i_i.R3\_ControlCoupling + \sum_{i=1}^{n} i_i.R2\_StampCoupling +$$

$$\sum_{i=1}^{n} i_i.R1\_DataCoupling - \sum_{i=1}^{n} c_i + 2.$$

$$\mathbf{V}(S.Complexity) = \mathbb{N}.$$

Attribute: Size Equivalent source lines of code (ENLOC) is a size measure which uses a gearing factor to get a size measure which allows size comparison between applications written in different programing languages. If $F = \{f_1, ..., f_n\}$ is a list of application functions, S is an application service and $F \subseteq S.realizedBy$, then

$$f(S.ENLOC) = \sum_{i=1}^{n} \frac{S.GearingFactor}{f_i.assignee.GearingFactor} * f_i.assignee.NLOC.$$

$$\mathbf{V}(S.ENLOC) = \mathbb{R}.$$

Attribute: GearingFactor. If S is an application service, then $\mathbf{V}(S.GearingFactor) = \mathbb{N}$. The gearing factor is given as evidence in the model.

Attribute: InternalCouplingAVG The InternalCouplingAVG is the internal average coupling of the application service. It is calculated as the arithmetic mean of the Fenton and Melton Software Metric [22] for all pair wise coupling measures

within the application service divided by the number of pairs. If $I = \{i_1, ..., i_n\}$ is a list of application collaborations, S is an application service, C is a application component where $C \subseteq S.realize$ and $I \subseteq S.realize.collaboration$, then

$$C.CouplingAVG = \frac{1}{n} \sum_{j=1}^{n} i_j.couplingInPair().$$

$$\mathbf{V}(C.CouplingAVG) = \{x \in \mathbb{R} : 0 \leq x < 6\}.$$

Attribute: InternalCouplingMAX The InternalCouplingMAX is the internal max coupling of the application service is. It gives the maximum value of all the connections pair to the application component within the application service. If $I = \{i_1, ..., i_n\}$ is a list of application collaborations, S is an application service, C is a application component where $C \subseteq S.realize$ and $I \subseteq S.realize.communication$, then $C.couplingMAX = couplingInPair(m)$ where $m \in P$ and $couplingInPair(i_j) \leq couplingInPair(m)$ for all elements in P. $\mathbf{V}(C.CouplingMAX) = \{x \in \mathbb{Q} : 0 \leq x < 6\}$.

Application Component. The application component class contains the attributes Coupling, Size, and Gearing Factor.

Attribute: ExternalCouplingAVG is calculated as the arithmetic mean of the Fenton and Melton Software Metric [22] for all pair wise coupling measures divided by the number of pairs. If $I = \{i_1, ..., i_n\}$ is a list of application collaborations, C is a application component and $I \subseteq C.collaboration$, then

$$C.CouplingAVG = \frac{1}{n} \sum_{j=1}^{n} i_j.couplingInPair().$$

$$\mathbf{V}(C.CouplingAVG) = \{x \in \mathbb{R} : 0 \leq x < 6\}.$$

Attribute: ExternalCouplingMAX gives the maximum value of all the connections pair to the application component. If $I = \{i_1, ..., i_n\}$ is a list of application collaborations, C is a application component and $I \subseteq C.communication$, then $C.couplingMAX = couplingInPair(m)$ where $m \in P$ and $couplingInPair(i_j) \leq couplingInPair(m)$ for all elements in P. $\mathbf{V}(C.CouplingMAX) = \{x \in \mathbb{Q} : 0 \leq x < 6\}$.

Size is measured as the number of non-commented lines of code (NLOC). If C is an application component, then $\mathbf{V}(C.NLOC) = \mathbb{N}$. The number of non-commented lines of code is given as evidence in the model.

Attribute: GearingFactor. If C is an application component, then $\mathbf{V}(C.GearingFactor) = \mathbb{N}$. The gearing factor is given as evidence in the model.

Application Collaboration. The application collaboration class contains the attribute Coupling (five different types).

If I is an application interaction, X and Y are both application components, and $\{X, Y\} \subseteq I.communicates$, then they have a:

Attribute: R5_ContentCoupling relation if X refers to the internals of Y, i.e., it branches into, changes data, or alters a statement in y.
$\mathbf{V}(R5\_ContentCoupling) = \mathbb{N}$.

Attribute: R4_CommonCoupling relation if X and Y refer to the same global variable. $\mathbf{V}(R4\_CommonCoupling) = \mathbb{N}$.

Attribute: R3_ControlCoupling relation if X passes a parameter to Y that controls its behavior. $\mathbf{V}(R3\_ControlCoupling) = \mathbb{N}$.

Attribute: R2_StampCoupling relation if X passes a variable of a record type as a parameter to Y, and Y uses only a subset of that record.
$\mathbf{V}(R2\_StampCoupling) = \mathbb{N}$.

Attribute: R1_DataCoupling relation if X and Y communicate by parameters, each one being either a single data item or a homogeneous set of data items that does not incorporate any control element. $\mathbf{V}(R1\_DataCoupling) = \mathbb{N}$.

The number of content, common, control, stamp, and data couplings are given as evidence in the model.

5 Case Study and Results

The revised metamodel has been applied in a case study conducted at the division for Student Records and Information Systems (VoS) at KTH – the Royal Institute of Technology, a university in Sweden. VoS is part of the university administration, their main responsibility is to provide system support for administration of courses, students and related processes. In the case study, nine application services governed by VoS have been modeled and analyzed with the metamodel. Three out of nine application services investigated are developed and maintained by VoS, two are commercially available of-the-shelf (COTS) products that are maintained by VoS, and the remaining four are developed and maintained by external service providers.

The information was collected through interviews and written correspondence. Interviews were also performed to validate the models created during the case study. The models were instantiated in the EAAT in order to make use of the P2AMF analysis language. Figure 5 shows four of the nine application services modeled. The color of the icon next to the application service name indicates the level of modifiability ranging from red to green, where green indicates high modifiability.

The nine application services modeled and analyzed indicate that the P2AMF analysis formalism is applicable to the architectural view of TEAMATe. The models also indicate that it is possible to apply the new metrics with TEAMATe. Further, the case study shows that it can be difficult to obtain coupling information regarding application component collaboration. Whereas size, programming language (for gearing factor), and complexity are easier to obtain. The complexity metric is structurally derived from the model requiring no extra effort once the model has been created. Programming language is known by the application developer and most development tools seem to have the functionality to count non-commented lines of code.

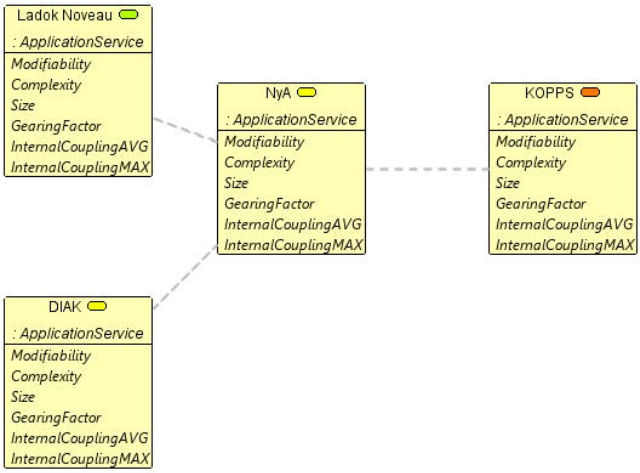

Fig. 5. A screenshot from EAAT showing four related application services. The color indicator next to the application service name illustrates the modifiability level.

Table 3. Results for the application services

Application service	Complexity	Coupling MAX	Size	Programming language	Modifiability
Alumni Community	3	3.6	23.807	C#	6
DIAK	4	3.6	N/A	ASP	9
KOPPS	7	4.6	35.156	Java	4
LadokPing	4	4.5	45.266	Java	4
LpW	6	4.9	178.758	N/A	5
Mina Sidor	10	4.6	11.323	Java	3
Nouveau	2	N/A	N/A	C++	12
NyA	0	3.6	819.341	Java	8
TimeEdit	3	4.6	N/A	C++	9

Since this paper and case study only focus on the architectural viewpoint of TEAMATe it is not as obvious how these models support decision-making, compared to when employing all viewpoints of TEAMATe including the change management and project specific classes which would provide the decision maker with information regarding the predicted cost of changing an architecture. However, only using the architectural view provides information that can be used when comparing different architecture scenarios. Also, if it is known that an application service has a modifiability value lower than others due to its complexity or high coupling, one might want to consider improving this service in order to prevent future change projects from becoming too costly. Or, when starting a new development project the information in these models can provide important input for risk analysis. E.g. if planning a change in one component, the modification's impact can be traced in the model in order to find what other components might be affected. This would prevent unknown ripple effects to occur.

6 Discussion

To measure the complexity of an IT system, the cyclomatic complexity by McCabe [17] has been used. The cyclomatic complexity is originally a software metric. Even though there are similarities of a control graph based on software source code and an architectural model displaying software components with their relations to one another, the suitability of the metric is not yet fully investigated within this context. [27] and [28] have both applied cyclomatic complexity to business process models (BPMs) indicating that the metric serves a good purpose providing general information about the complexity of a BPM.

The categorization used for the modifiability metric provides insight to whether the service is easy to modify or not. Neither the less it can be of guidance when evaluating possible architectural scenarios to one another. Here modifiability is based on the three attributes size, coupling, and complexity. These are not the only attributes that determines the modifiability of an application service. E.g. in the Definition and Taxonomy for Software Maintainability presented in [11] 140 factors are included (however some of these are related to the management process and the team involved).

The use of TEAMATe when making decisions relating to enterprise IT is beneficial due to its ability to locate potential risks concerning a system architecture. Employing TEAMATe raises awareness of the enterprise system including an understanding of how the different applications relate to each other.

Aligning TEAMATe with ArchiMate might be good for a majority of companies. However, other companies might already employ a different metamodel for their EA initiative. For the specific modeling project alignment with other metamodels might be necessary.

The Enterprise Architecture Analysis Tool is currently a research product more than a commercially available software. Hopefully it will soon be user-friendly enough for non-academic use. Unfortunately, there are no other tools available today that support the suggested type of architecture analysis.

7 Conclusions

It is feasible to combine EA models with modifiability analysis to provide support when making decisions about IT systems and their implementation. This paper has provided insight on how the enterprise architecture modeling language ArchiMate can be aligned with The Enterprise Architecture Modifiability Analysis Tool (TEAMATe). Further, well-known software metrics have been incorporated in TEAMATe in order to improve the analysis capabilities of TEAMATe regarding architecture modifiability. An improvement made possible by changing formal analysis language from the Probabilistic Relational Models and Bayesian networks to the Predictive, Probabilistic Architecture Modeling Framework (P2AMF). These improvements have been tested in a case study, where the modeling and analysis were done with the Enterprise Architecture Analysis Tool (EAAT), with good results.

7.1 Future Work and Implications

To the industrial practitioner, the present paper assists the enterprise modeling effort when the concern is focused on modifiability of application services. If the suggested metamodel is employed one can compare different architecture scenarios, find architectural improvement possibilities, and highlight risks early in change projects.

To the EA tool industry, the present paper hints to potential new features or products. Tools incorporating the suggested analysis capabilities and containing common metrics for architectural evaluation would give a more quantitative support to the user's modeling effort.

To the scientific community, the presented metamodel combines the approach of enterprise architecture modeling with modifiability metric analysis. These two, previously separate, 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.

Of course, the different parts of TEAMATe can still be further improved. More metrics can be added. The change management viewpoint and the change project viewpoint can both be revised regarding metrics used, related modeling languages, and analysis formalism.

In order to expand this study to the broader EA context, future work will be conducted to incorporate the presented metamodel with other quality goals that are of importance when evaluating architectures. Such quality goals could be those presented in [29]; application usage, service availability, service response time, and data accuracy, as well as interoperability as presented in [25].

References

1. Ross, J., Weill, P., Robertson, D.: Enterprise architecture as strategy: Creating a foundation for business execution. Harvard Business Press (2006)
2. Department of Defense Architecture Framework Working Group: DoD Architecture Framework, version 1.5. Technical report, Department of Defense, USA (2007)
3. Zachman, J.A.: A framework for information systems architecture. IBM Systems Journal 26, 276–292 (1987)
4. The Open Group: The Open Group Architecture Framework (TOGAF) - version 9. The Open Group (2009)
5. Lankhorst, M.: Enterprise architecture at work: Modelling, communication and analysis. Springer-Verlag New York Inc. (2009)
6. Kurpjuweit, S., Winter, R.: Viewpoint-based meta model engineering. In: Enterprise Modelling and Information Systems Architectures, EMISA 2007 (2007)
7. Lagerström, R., Johnson, P., Ekstedt, M.: Architecture analysis of enterprise systems modifiability – a metamodel for software change cost estimation. Software Quality Journal 18, 437–468 (2010)
8. Lagerström, R., Johnson, P., Höök, D.: Architecture analysis of enterprise systems modifiability – models, analysis, and validation. Journal of Systems and Software 83(8), 1387–1403 (2010)
9. Boehm, B., Madachy, R., Steece, B., et al.: Software Cost Estimation with Cocomo II with Cdrom. Prentice Hall PTR (2000)

10. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, 2nd edn. Addison-Wesley Longman Publishing Co., Inc., Boston (2003)
11. Oman, P., Hagemeister, J., Ash, D.: A definition and taxonomy for software maintainability. Technical report, Software Engineering Lab (1992)
12. IEEE Standards Board: IEEE standard glossary of software engineering technology. Technical report, The Institute of Electrical and Electronics Engineers (September 1990)
13. Halstead, M.: Elements of Software Science. Operating and programming systems series. Elsevier Science Inc. (1977)
14. Laird, L., Brennan, M.: Software measurement and estimation: a practical approach, vol. 2. Wiley-IEEE Computer Society Pr. (2006)
15. Henry, S., Kafura, D.: Software structure metrics based on information flow. IEEE Transactions on Software Engineering SE-7(5), 510–518 (1981)
16. Frappier, M., Matwin, S., Mili, A.: Software metrics for predicting maintainability. Software Metrics Study: Tech. Memo 2 (1994)
17. McCabe, T.: A complexity measure. IEEE Transactions on Software Engineering (4), 308–320 (1976)
18. Park, R.: Software size measurement: A framework for counting source statements. Technical report, DTIC Document (1992)
19. Curtis, B., Krasner, H., Iscoe, N.: A field study of the software design process for large systems. Communications of the ACM 31(11), 1268–1287 (1988)
20. Maraia, V.: The Build Master: Microsoft's Software Configuration Management Best Practices. Addison-Wesley Professional (2005)
21. Jones, C.: Applied software measurement: assuring productivity and quality. McGraw-Hill, Inc. (1991)
22. Fenton, N., Melton, A.: Deriving structurally based software measures. Journal of Systems and Software 12(3), 177–187 (1990)
23. OMG: Object constraint language, version 2.2. Technical report, Object Management Group, OMG (February 2010)
24. Akehurst, D., Bordbar, B.: On Querying UML Data Models with OCL. In: Gogolla, M., Kobryn, C. (eds.) UML 2001. LNCS, vol. 2185, pp. 91–103. Springer, Heidelberg (2001)
25. Ullberg, J., Franke, U., Buschle, M., Johnson, P.: A tool for interoperability analysis of enterprise architecture models using Pi-OCL. In: Enterprise Interoperability IV, pp. 81–90 (2010)
26. IEEE: IEEE recommended practice for architectural description of software-intensive systems. Technical report, Technical Report IEEE Std 1471-2000. IEEE Computer Society (2000)
27. Cardoso, J., Mendling, J., Neumann, G., Reijers, H.: A Discourse on Complexity of Process Models. In: Eder, J., Dustdar, S. (eds.) BPM Workshops 2006. LNCS, vol. 4103, pp. 117–128. Springer, Heidelberg (2006)
28. Gruhn, V., Laue, R.: Complexity metrics for business process models. In: 9th International Conference on Business Information Systems (BIS 2006), Citeseer, vol. 85, pp. 1–12 (2006)
29. Närman, P., Buschle, M., Ekstedt, M.: An enterprise architecture framework for multi-attribute information systems analysis. Systems and Software Modeling (accepted to be published, 2012)