

An Enterprise Architecture Management Pattern for Software Change Project Cost Analysis

Robert Lagerström, Pontus Johnson, David Höök

Industrial Information and Control Systems
the Royal Institute of Technology
Osquidas väg 12
10044 Stockholm
{robertl, pj101, davidh}@ics.kth.se

Abstract: Business environments today progress and change rapidly. Most business processes are supported by information systems and since the business processes change the information systems need to be changed as well. An essential issue with today's software systems is that many of them are interconnected, thus a change to one system may cause a ripple effect among other systems. Also, numerous systems have been developed and modified during many years and to make further changes to them requires a lot of effort from the organization. This paper suggests enterprise architecture as an approach to model software systems and their environment. An enterprise architecture management pattern in the form of a metamodel for change project costs modeling and analysis is presented. IT decision makers can use this metamodel in order to make cost predictions and do risk analysis for their change projects. The employment of the model is illustrated with ten projects.

1 Introduction

Enterprise architecture is an approach to enterprise information systems management that relies on models of the information systems and their environment. Instead of modifying the enterprise information system using trial and error, a set of models is proposed to predict the behavior and effects of changes to the system. The enterprise architecture models allow reasoning about the consequences of various scenarios and thereby support decision making. What constitutes a "good" enterprise architecture model is contingent on the purpose the model is intended to fill, that is what type of analysis it will be subjected to. For instance in the case of analyzing software maintainability, here defined as the cost of change, it is necessary that the models contain information on maintainability, e.g. component source code understandability, documentation quality, change management process maturity etc. [Jo07]

Ernst [Er08] proposes patterns as an approach to manage enterprise architecture. A pattern is a general, context dependent, reusable solution to a common problem. The enterprise architecture management (EAM) patterns are employed to document different solutions to reoccurring enterprise architecture problems. Ernst discusses three types of EAM patterns; Methodology Patterns, Viewpoint Patterns, and Information Model Patterns. This paper focuses on an Information Model Pattern (I-Patterns). The I-Pattern proposed in this paper documents an enterprise architecture metamodel for software maintainability analysis. Describing and documenting patterns is usually done following one of the suggested pattern forms, the I-Pattern description in this paper is influenced by the *CompactForm* structure: context, problem, forces, solution, and resulting context [Tr09].

The purpose of this paper is to present an enterprise architecture management pattern in the form of a metamodel for maintainability, i.e. change cost, analysis. The metamodel presented in this paper is a type of metamodel called abstract model - a metamodel integrated with a Bayesian network, c.f. section 3. [La07]

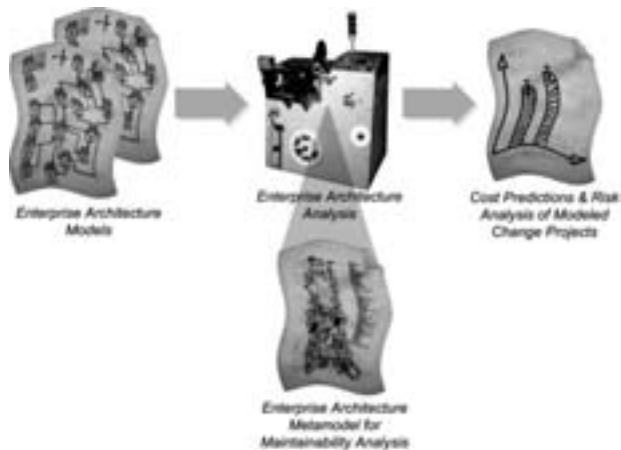


Figure 1. The enterprise architecture analysis approach.

The abstract model presented in this paper consists of entities, e.g. system or developer, and attributes, e.g. maintainability and understandability. Attributes are connected with other attributes via causal relations. Adding information to one attribute makes it possible to calculate the value of other attributes, due to the inherited Bayesian formalism. For example adding information about a component's source code coupling helps predicting the component's source code understandability.

2 Related Work

A number of enterprise architecture initiatives have been proposed, for example The Open Group Architecture Framework (TOGAF) [Op05], the Zachman Framework [Za87], Enterprise Architecture Planning (EAP) [SH92], and the General Enterprise Reference Architecture and Methodology (GERAM) [If99]. There are also numerous metamodels proposed for enterprise architecture modeling, e.g. the ones presented by Cummins [Cu02], O'Rourke [OFS03], Lankhorst [La05], and Niemann [Ni06]. Considering the suitability of the frameworks and metamodels for the purpose of maintainability analysis, it is found that a number of the proposed metamodels are not detailed enough to provide the information required for our analysis purpose. When analyzing maintainability it is necessary that the metamodel is capable to model systems, components, documentation, tools, processes, developers etc. and information such as component coupling and size, developer expertise, change management process maturity etc. Few metamodels hold any of this information, none have it all.

Several methods for analyzing maintainability and change costs have been suggested, for example the CONstructive COSt MOdel (COCOMO) for software maintenance cost estimation [Bo81], the Definition and Taxonomy for Software Maintainability [Om92], the maintainability part of the Quality Model proposed by ISO/IEC [Is01], the maintainability software metrics suggested by Fenton and Pfleeger [FP97], and the Software Architecture Analysis Method (SAAM) concerning modifiability [BCK98]. Existing enterprise architecture frameworks and metamodels don't contain the proper information for maintainability analysis and that previously proposed maintainability measures haven't been incorporated in any enterprise architecture metamodel, we propose such a metamodel in this paper.

3 Introduction to Enterprise Architecture Analysis

An abstract model is an enterprise architecture metamodel containing entities and entity relations, augmented with attributes and attribute relations. Entities represent the objects of interest when modeling, e.g. systems, components, persons, or processes. Entity relations connect two entities, e.g. "documentation *describes* system" or "person *is a resource of* process". Entity relations also state the multiplicity of the relationship between the entities, c.f. Figure 2.

Attributes of an abstract model represent variables related to the entities. Here attributes are discrete random variables that may assume values from a finite domain, e.g. {true, false}, {high, medium low} or {0-9, 10-99, 100-999}. The attributes represent concepts such as maintainability, complexity, quality of documentation, number of lines of code, cost, or expertise. Attributes of an abstract model relate to other attributes with causal relations. The causal relation indicates a probabilistic dependence between two attributes, in the same way as the causal relations in Bayesian networks. This type of relation is graphically represented as a grey arrow, c.f. Figure 2. The probability of the values of a target attribute is thus dependent on the values of its source attributes. This is represented in conditional probability matrices, for example the attribute component complexity affected by component size can be represented as the conditional probability matrix in Table 1.

Table 1. A conditional probability matrix example.

Component Size		>1.000.000	999.999-100.000	99.999-0
Component Complexity	High	0.75	0.25	0.05
	Medium	0.20	0.50	0.20
	Low	0.05	0.25	0.75

One entry in the matrix is $P(Complexity=High/Size=>1.000.000)=0.75$, which means that the probability of the attribute complexity assuming the value *high* given that size is larger than one million is 75 %. The conditional probabilities are an important factor in the presented approach of performing analysis using enterprise architecture models. Thus, based on information on some attributes an expected value of other attributes can be calculated.

If the probabilities of the source attributes are known, it is possible to infer a value for the target attribute using the law of total probability,

$$P(A) = \sum_i P(A|B_i)P(B_i)$$

Also, using Bayes' rule,

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

makes it possible to calculate the values of source attributes based on the probabilities of a target attribute [Je01].

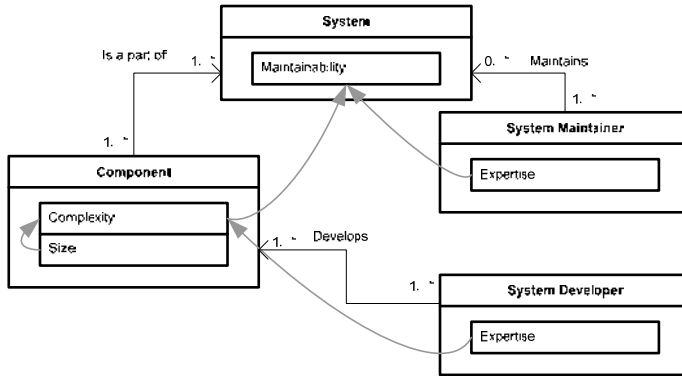


Figure 2. A small abstract model example.

Let's assume we use the exemplifying abstract model in Figure 2 to evaluate the maintainability of an automatic meter reading (AMR) system. The first thing we need to do is to collect information about the system, the abstract model tells us what information we need to find. When the information has been collected we specify it in the model, thereby creating an instantiation of the model. These instantiations are called concrete models. When a concrete model of our system and its environment has been created we can use the Bayesian mathematics to calculate the maintainability of the system. The AMR system has three software components: Meter reading, Outage management, and Calculate energy cost. There are two persons involved in the system development, Peter and Ann. Peter develops the Meter reading component, and Ann develops the Outage management and Calculate energy cost components. There is also a person at the company doing software maintenance, John. The concrete model is visualized in Figure 3.

The next step is to collect information about the attributes in the concrete model, e.g. the size of the components and the expertise of the personnel. This can be done by conducting interviews or workshops with people working with the system under analysis, or one can go through relevant documents, such as design specifications. In this case information about the size of the components can be found in the source code using standard tools for source code size measures. When the attribute values have been collected they should be entered into the model. Once the model has information on some attributes it is possible to calculate the expected values of other attributes. E.g. when we enter component size, developer expertise, and maintainer expertise into the model, we can calculate the values of component complexity and system maintainability, c.f. Figure 3.

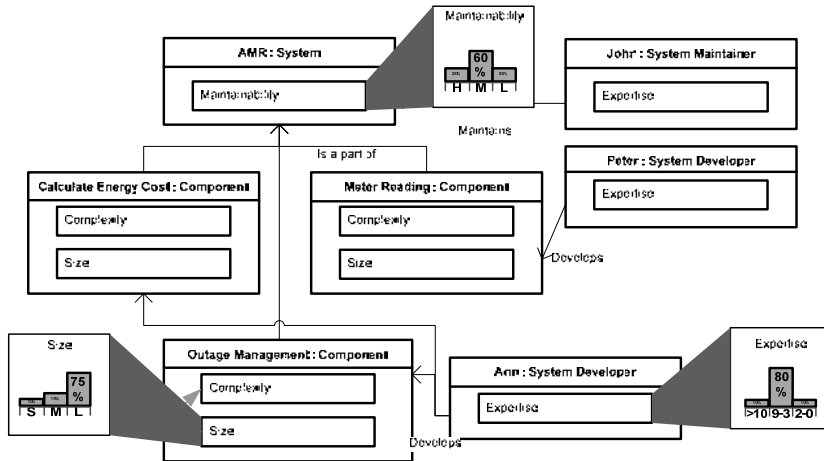


Figure 3. A small concrete model example.

4 The software change project cost analysis pattern

The software change project cost analysis pattern following the *CompactForm* [Tr09] in the current paper is built up using the following lynchpins; *Problem* for providing the background to the real world in which the pattern have been identified, *Forces* for pointing out what aspects of the pattern one have to concern in order to have the abstract model empirically adoptable, in this case the effort to model traded against the accuracy of the model, as well as the computational capabilities of the model, *Context* for setting up the empirical environment in which the metamodel can be used in, in this case in decision making situations in enterprise wide system software change projects, *Solution* – the complete abstract model with all relevant entities, attributes and attribute relations, and *Known Uses* which includes a presentation of ten projects and the analysis results of these.

4.1 Problem Description and Forces

Since business processes needs to change with changing business environments the supporting information systems needs to be changed as well. These modifications can vary from adding a functional requirement in one system to implementing a service oriented architecture for the whole enterprise. Today's software systems are interconnected to a large extent, thus a change in one system may cause a ripple effect among other systems. The systems have also been developed and modified during many years on an ad hoc basis and making that further changes to them requires a lot of effort.

Changes in enterprise systems are typically executed in projects and IT decision makers often find it difficult to predict and plan their change projects. Thus, a large proportion of the projects with the purpose of changing a software system tend to take longer time and cost more than expected. This may often occur due to lack of information about the affected systems. Therefore, it is useful for IT decision makers to be able to analyze how much effort a certain change to an enterprise information system would require.

What kind of metamodel is needed to support software change project cost analysis?

The main forces influencing the solution are (1) **minimal modeling effort vs. prediction accuracy** and (2) **the need of a formal analysis engine**, for the sake of ease of computational analysis.

4.2 Context

The pattern solution is intended to be employed in decision situations when prioritizing and choosing change projects. Models based on the presented metamodel enable the decision maker to: (1) make predictions of the maintainability for different software projects and (2) retrieve information for risk analysis, thus improving the support for planning and decision making in software change project management.

4.3 Solution

This subsection presents an enterprise architecture metamodel in the form of an abstract model, c.f. Figure 4. The model is to be employed in software system maintainability analyses for change project cost predictions.

The presented abstract model is influenced by and based on earlier suggested maintainability and change cost frameworks, scientific papers, and books, such as the work done by Boehm [Bo81], Oman et al. [Om92], ISO [Is01], Fenton [FP97], and Bass et al. [BCK98].

IEEE defines maintainability as the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment [Ie90]. We interpret ease as the cost of making the modification. Instead of focusing on a single software system or component, as many have done before, this metamodel intends to support maintainability of enterprise wide systems of systems.

The abstract model focuses on modeling the systems and its surrounding environment involved in or affected by the modifications implemented in a change project. The main attribute of the abstract model is the cost attribute in the entity change project enabling predictions of the cost of a specific change project. Cost is here measured as the number of man hours needed to make the change. The abstract model for change project cost predictions is presented in Figure 4. A complete description of the entities and attributes of the abstract model can be found in [LJ08].

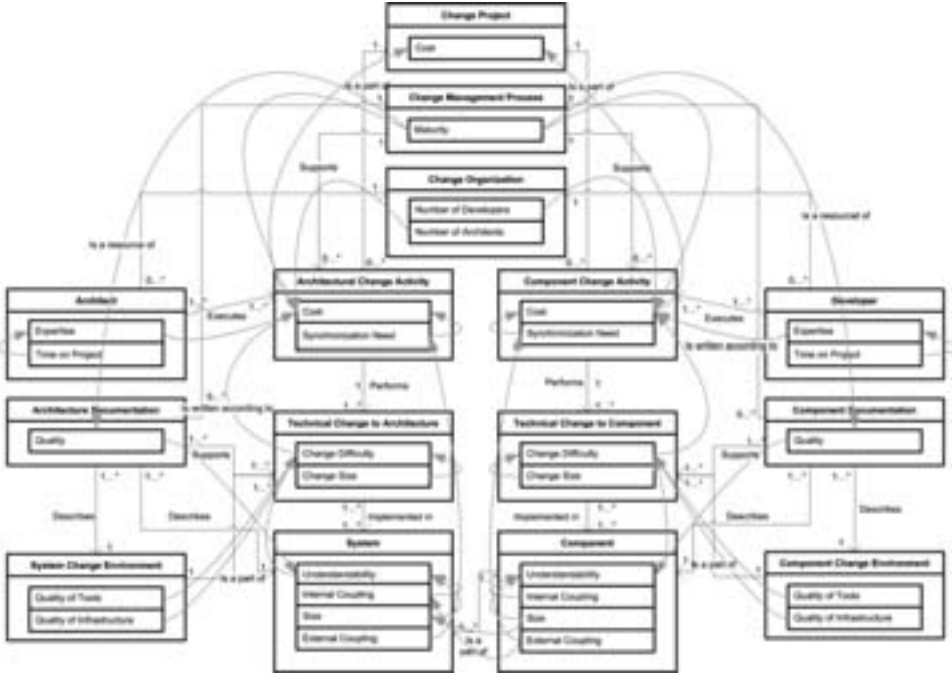


Figure 4. The abstract model for maintainability.

The attributes in the model, c.f. Figure 4, are all associated with a conditional probability matrix. The conditional probabilities in each matrix are based on the knowledge from 43 experts in the field of enterprise system change management. This information was collected by surveying industrial and academic experts on how the different attributes of the presented abstract model affect each other. An example of a typical question is “How much does the change management process maturity affect the architectural change activity cost?”. The alternatives for answering were “High”, “Low”, “None” or “I don’t know”. Our algorithms for Bayesian probability learning are based on the number of answers on each alternative for each question (excluding “I don’t know” answers). Every alternative has its own weight for modeling uncertainties in the variable relations. All answers are summed up and normalized for each individual question in consistency with the equation presented below.

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

$$i \in [1,2,3], \quad j \in [1,2,3]$$

Here n represents the question number; i and j the ordinal choices of correlation; and z the number of answers. In this specific 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 variable affecting the outcome then there will be a joint probability relation. A representation of the joint probability between questions $1 \rightarrow m$ is represented as $P(Y/X_1, \dots, X_m)$. For each answer alternative i with relating output correlation j the joint probability is calculated as

$$P(y_j | x_{i,1}, \dots, x_{i,m}) = \sum_{n=1}^m \frac{P(y_j | x_{i,n})}{m}$$

The outcome of the presented expert elicitation method is complete conditional probability matrices for all attributes in the abstract model. [La09]

4.4 Known Uses

In section 3 a short example was provided in order to show how abstract models can be used and how abstract models turn into concrete models. This section illustrates the presented maintainability abstract model's capability of predicting the cost order of change projects using two different case studies. In the first case eight projects, C-J, within a large Nordic manufacturing company was studied. The second case considered two projects, A-B, conducted within a Nordic consultancy firm. In the subsequent subsections we first present Project F in more detail to show how the abstract model can be employed and secondly the actual predictions of the cost order for the ten projects are listed, and compared to the actual cost order of the projects, as can be seen in Table 2.

Project F was initiated since the company predicted increased sales and thus an increased amount of employed products. To still be able to manage the sold products a new function in the Product Management Portal was needed. In addition to this, a more secure, redundant and scalable server environment was needed to be set up along the development of the new software application. The project had several objectives concerning improvement of the product management business, namely; to reduce the amount of hours spent on administration related to product management, to improve the support for the distributors of the products, to improve the quality of current services, to obtain a scalable, redundant and secure communication infrastructure and to future proof the communication and server environment in terms of the functionality they will be able to incorporate.

In Figure 5 an excerpt of the concrete model for project F is depicted with some of the collected data visualized as probability distributions over the states of the concerned attribute's quantitative scale. Project F's main resources were comprised by ten software developers and three system architects. The developer team had an experience of between 2 and 4 years working with the fleet management system, an experience between 0 and 1 year working in software change projects as well as working with the specific software design language employed in the project. The architect team was a bit more experienced in their domain. They had over 4 years of experience within both the system environment, within change projects in general as well as of using the design specific language. The developers and the architects spent between 81 and 100 percent respectively between 0 and 20 percent of their available time in Project F.

The change management process within the company got the lowest level of maturity possible. This can be deduced to the company's low values on document maturity, metrics maturity and activities maturity as well as their low degree of maturity concerned with assigning responsibilities within the change management process.

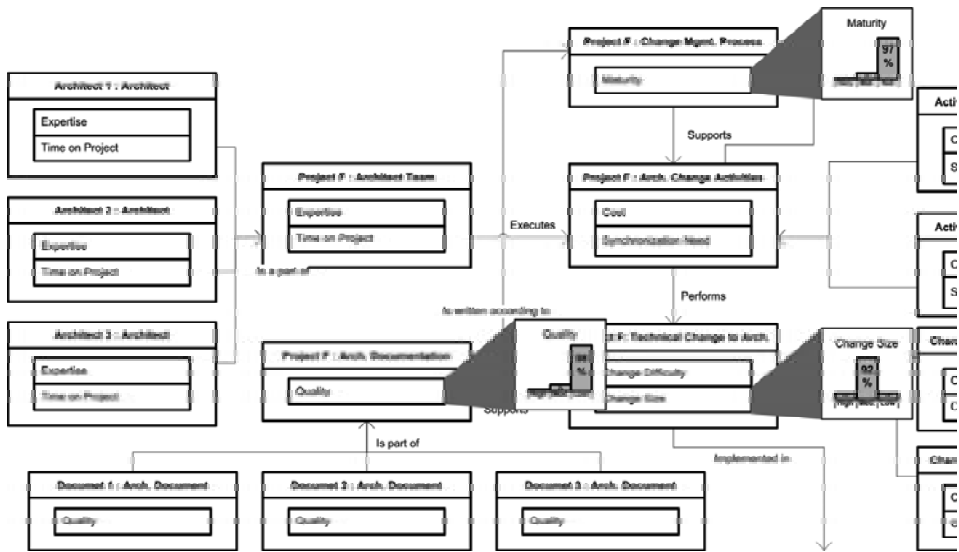


Figure 5. Excerpt of the concrete model utilized for predicting the change costs in Project F

In below Table 2 we can see how the cost predictions for the ten projects turned out in comparison to the actual cost.

Table 2. A magnitude comparison of predicted and actual costs of the ten studied projects.

	<i>Predicted</i>	<i>Actual</i>
1	A	A
2	F	F
3	B	H
4	D	B
5	J	J
6	H	E
7	E	C
8	I	I
9	C	D
10	G	G

Totally, five projects are predicted at the same cost level as the actual outcome and three projects are only one or two spots away from the actual outcome. Only two projects were predicted more than two spots away from their outcome. This indicates that the suggested metamodel suits its purpose well, but also that there is room for improving its assessment capabilities using the method put forward in section 4.3 with more input data.

5 Summary and Outlook

This paper presented an enterprise architecture management pattern in the form of a metamodel for enterprise system maintainability analysis. The metamodel consists of entities with accompanying attributes that can be used to create enterprise architecture models. The information contained in these models support quantitative analysis of maintainability, i.e. the change cost of different projects. IT decision makers employing the model will be able to make predictions of the change cost order for different software projects and retrieve information for risk analysis in these change projects. Thus, support for prioritizations and for choosing change projects in a project portfolio are enabled.

6 Acknowledgements

The authors of this paper would like to thank Kennet Namini and Najib Mirkhani for participating in the data collection phase of the case studies, Johan König for his work in the expert elicitation phase of the project, all the participants of the workshops and surveys conducted, the reviewers of this paper whose comments improved the quality of the paper, and finally our paper shepherd Alexander Ernst who significantly increased the value of the paper contribution to the pattern community.

References

- [BCK98] L. Bass, P. Clements and R. Kazman, Software Architecture in Practise, the SEI Series in Software Engineering, Addison Wesley Longman, 1998.
- [Bo81] B. Boehm, Software Engineering Economics, Prentice Hall, 1981.
- [Cu02] F. Cummins, Enterprise Integration – An Architecture for Enterprise Application and Systems Integration, John Wiley & Sons, 2002.
- [Er08] A. Ernst, Enterprise Architecture Management Patterns, Pattern Languages of Programs Conference 2008 (PLoP08), Nashville, 2008.
- [FP97] N. Fenton and S. Pfleeger, Software Metrics – A Rigorous & Practical Approach, Second Edition, PWS Publishing Company, 1997.
- [Ie90] IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990, ISBN 1-55937-067-X, 1990.
- [If99] IFAC-IFIP, “Task Force on Architectures for Enterprise Integration Geram - Generalized enterprise reference architecture and methodology”, Version 1.6, 1999.
- [Is01] ISO/IEC 9126-1, Software Engineering – Product Quality, 2001.
- [Je01] F. Jensen, Bayesian Networks and Decision Graphs, Springer-Verlag, 2001.
- [Jo07] P. Johnson, R. Lagerström, P. Närman, and M. Simonsson, Enterprise Architecture Analysis with Extended Influence Diagrams, Information Systems Frontiers, Vol. 9, No 2, May 2007.
- [La05] M. Lankhorst et al., Enterprise Architecture at Work – Modelling, Communication, and Analysis, Springer-Verlag, 2005.
- [La07] R. Lagerström, Analyzing System Maintainability using Enterprise Architecture Models, Journal of Enterprise Architecture, November 2007.
- [La09] R. Lagerström, P. Johnson, D. Höök, and J. König, Software Change Project Cost Estimation – A Bayesian Network and a Method for Expert Elicitation, Accepted to the Software Quality and Maintainability (SQM) workshop, 2009.
- [LJ08] R. Lagerström and P. Johnson, Using Architectural Models to Predict the Maintainability of Enterprise Systems, In Proceedings of the 12th European Conference on Software Maintenance and Reengineering, April 2008.
- [Ni06] K. Niemann, From Enterprise Architecture to IT Governance – Elements of Effective IT Management, Friedr. Vieweg & Sohn Verlag, 2006.
- [OFS03] C. O’Rourke, N. Fishman, and W. Selkow, Enterprise Architecture – Using the Zachman Framework, Thomson Learning, 2003.
- [Om92] P. Oman, J. Hagemester and D. Ash, “A Definition and Taxonomy for Software Maintainability”, SETL Report #91-08 TR, University of Idaho, Moscow, ID 83843, 1992.
- [Op05] The Open Group, “The Open Group Architecture Framework”, Version 8, 2005.
- [SH92] S.H. Spewak and S.C. Hill, Enterprise Architecture Planning - Developing a Blueprint for Data, Applications and Technology, John Wiley and Sons, 1992.
- [Tr09] Ben Tremblay, CompactForm, <http://c2.com/cgi/wiki?CompactForm>, 24 Feb. 2009.
- [Za87] J. Zachman, “A framework for information systems architecture”, IBM Systems Journal, 26(3), 1987.