Concern-Oriented Analysis and Refactoring of Software Architectures using Dependency Structure Matrices

Bedir Tekinerdoğan
Bilkent University
Department of Computer Engineering
06800 Bilkent, Ankara, Turkey
bedir@cs.bilkent.edu.tr

Frank Scholten, Christian Hofmann, Mehmet Aksit
Department of Computer Science,
University of Twente,
P.O. Box 217 7500 AE Enschede, The Netherlands
{f.scholten | c.hofmann | aksit}@cs.utwente.nl

ABSTRACT

Current scenario-based architecture analysis methods analyze the architecture with respect to scenarios that relate to stakeholder concerns. Albeit the primary motivation is to analyze the impact of stakeholders' concerns, it appears that concerns are not explicitly represented as first class abstractions. The lack of an explicit notion of concern in scenario-based analysis approaches can result in an incomplete analysis because scenarios are too specific and can only partially represent the concerns. We propose the concern-oriented architecture analysis method (COSAAM) that builds on scenario-based approaches but includes an explicit notion of concern in the analysis. COSAAM applies Dependency Structure Matrices (DSMs) to represent and analyze the dependencies among scenarios, concerns and architectural elements. Further, COSAAM extends DSMs by introducing explicit DSM patterns and heuristic rules for analyzing the impact of concerns on the architecture and for supporting the refactoring of the architecture.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques

General Terms: Design, Documentation, Languages.

Keywords: Concern-Oriented Modeling, Dependency Structure Matrix, Software Architecture Analysis

1. INTRODUCTION

Software architecture forms one of the key artifacts in the entire software development life cycle since it embodies the earliest design decisions and includes the gross-level components that directly impact the subsequent analysis, design and implementation [1]. Accordingly, it is important that the architecture design supports the software system qualities required by the various stakeholders. For ensuring the quality factors the common assumption is that identifying the fundamental concerns for architecture design is necessary and various architecture design methods have been introduced for this purpose. To verify that the right concerns have been identified usually static analysis of formal architectural models is applied or a set of architecture analysis methods as described in [2] are adopted. Very often scenario-based architecture analysis methods are applied [2]. In general, these analysis methods take as input the architecture design and measure the impact of predefined scenarios on it in order to identify the potential risks and the sensitive points of the architecture. This helps to predict the quality of the system before it is built, thereby reducing unnecessary maintenance costs.

Although, the key motivation is to analyze the architecture with respect to the concerns it appears that concerns are not explicitly represented as first class abstractions in scenario-based analysis approaches. This is somehow surprising since the primary motivation for analyzing the architecture is in fact the analysis of the stakeholder concerns. In general a concern is defined implicitly in the scenarios and the evaluation of the architecture is performed for scenarios. However, very often not the particular scenario but the concern that is addressed by the scenario is of importance. In this sense a concern might cover a broader set of scenarios. A concern is generally defined as any matter of interest that is relevant to a stakeholder. The lack of an explicit representation of concerns in the architectural analysis reduces the understandability and traceability of the impact of concerns. An explicit insight in the concerns and their impact on the architecture is not only necessary for the impact analysis but also for the refactoring process that utilizes the results of the analysis process to enhance the architecture.

We propose the concern-oriented architecture analysis method (COSAAM) that builds on existing scenario-based architecture analysis methods. For representing and analyzing concerns we use dependency structure matrices (DSMs) [8][10][3]. DSMs can be used to analyze the properties of complex applications. In DSM-based architectural analysis in particular the coupling between the architectural models are depicted and an optimal decomposition is aimed by reducing the couplings using predefined matrix operations. COSAAM consists of three basic processes. In the preparation phase scenarios are elicited. Based on clustering mechanisms in DSMs we derive a number of concerns. In the analysis phase together with the architectural elements, the extracted concerns are represented in a so-called *Domain Mapping Matrix (DMM).* Together with DMM we have defined a set of heuristic rules for analyzing the concerns. Finally, in the transformation phase the result of the analysis is used to redefine the architecture. One of the key contributions of COSAAM is that it defines explicit heuristics for supporting the DM-based analysis.

The remainder of this paper is organized as follows. In section 2 a running example, the design of a Window Management System will be described. Section 3 describes the steps of COSAAM using the case example. Section 4 presents the related work and finally section 5 presents the conclusions.

EA'09, March 3, 2009, Charlottesville, VA, USA.

2. COSAAM PROCESS

COSAAM is an iterative software architecture evaluation and transformation method that enhances scenario-based architecture analysis methods with DSM-based analysis. In particular COSAAM builds on the earlier Software Architecture Analysis Method (SAAM) [2] and Aspectual Software Architecture Analysis Method (ASAAM) [9]. ASAAM was built on SAAM to identify so-called aspectual scenarios and from these architectural aspects. While both SAAM and ASAAM analyze the architecture from a scenario perspective, COSAAM utilizes a concern-oriented approach and includes an explicit and systematic transformation step.

COSAAM consists of the three phases: *preparation*, *analysis* and *transformation*. The preparation phase establishes the artifacts used in the COSAAM evaluation: a candidate software architecture design and a collection of concerns of stakeholders. The analysis phase involves a characterization and measurement of scattering and tangling of concerns and modules. The information provided during this analysis is used in the transformation phase, in which the candidate software architecture is transformed. An iteration of COSAAM consists of all the activities of the analysis and transformation phases. In the following subsections we will elaborate on the three phases using an example case study.

2.1 Preparation Phase

In the preparation the basic inputs for the analysis are defined: a *candidate software architecture design* and a *collection of concerns* from stakeholders. The phase consists of two parallel activities: *describe candidate architecture* and *concern identification*, which are explained below.

2.1.1 Describe Candidate Architecture

To describe the architecture conventional software architecture modeling approaches are applied [1]. The architecture is then mapped to a DSM. Figure 1a shows for example the DSM for an example Window Management System architecture. The acronyms EM, PM, WM and SM represent *EventManager*, *ProcessManager*, *WindowManager* and *ScreenManager*, respectively. The rows and columns represent subsystems and the arrow represents dependencies between architectural components. In the figure we can observe, for example, that *EventManager* depends on *WindowManager*.

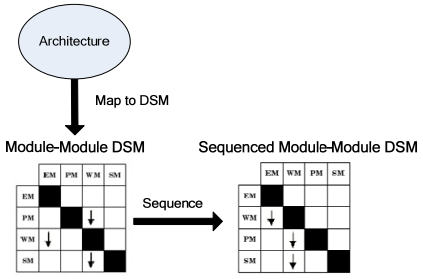

Figure 1. Sequencing DSM for WMS

Several predefined DSM operations can be used such as *partitioning, tearing, banding* and *clustering*, to optimize the decomposition [8]. This results in a so-called lower-triangular form which helps to reason about dependencies between modules. Figure 1 shows, for example the result of partitioning the DSM for WMS architecture.

2.1.2 Concern Identification

In the concern identification step we derive the concerns that are important for the stakeholders. For this two steps can be followed. First, concerns can be *reused* from existing projects, requirements specifications or domain models. Second, concerns can also be derived from scenarios developed by stakeholders. For the latter case we define a scenario-scenario DSM and derive concerns based on clustering of scenarios. The DSM clustering process is illustrated in Figure 2 in which we have derived 9 clusters of scenarios, i.e. 9 concerns.

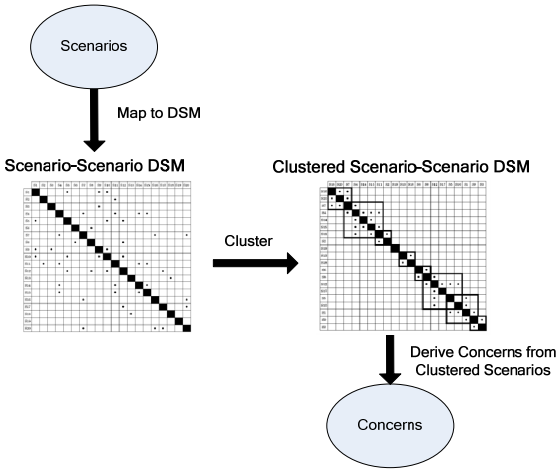

Figure 2. Scenario-Scenario DSM and Module-Module DSM

2.2 Analysis Phase

In the analysis phase of COSAAM the existing architecture is analyzed with respect to the given concerns. The analysis phase consists of the sub-phases *Initialize Concern-Module DMM, Characterize Concerns and* Modules and *Measure Impact of Concerns.* We explain these steps in the following subsections.

2.2.1 Initialization of Concern-Module DMM

The first step in the analysis phase is the mapping of concerns to modules. For this we apply the so-called *Domain Mapping Matrix (DMM).* In contrast to DSM's that represent the mapping of elements in the same domain DMMs represent the mapping between elements from different domains [10]. In COSAAM we use a DMM to show the mapping from concerns to architectural elements. Table 1 shows, for example, the mapping of concerns to elements of an architecture for the WMS case. Every concern can be directly or indirectly mapped to an architectural element. A direct mapping, represented by D in the table, means that the corresponding concern is implemented by the architectural element. An indirect mapping, represented by I, means that the architectural element needs to be changed to meet that concern. This is similar to the distinction between direct and indirect scenarios as it is defined in the SAAM. The difference here is that our abstraction is at the concern level, which clusters a set of scenarios. For example, in Table 1, concern *monitoring* concern has been evaluated as being indirect for all the modules.

Table 1. DMM for mapping concerns to Modules

Concerns \ Modules	Event Manager (EM)	Window Manager (WM)	Process Manager (PM)	Screen Manager (SM)
Monitoring (MO)	I	I	I	I
Portability (OP)	I	I	I	I
Failure Management (FM)	I	I	I	I
Process Term.(PT)	D	D	D	
Process Man. (PM)		D	D	
Device Man. (DM)	I			
Window Man.(WM)		D		
Appearance Conf. (WAC)		D		
Screen Man. (SM)				D

2.2.2 Characterize Concern and Modules

In the step *Characterize Architectural Modules* we analyze and characterize the architectural modules and concerns. The characterization follows the process as depicted in Figure 3. Here the rounded rectangles represent the different characterizations of selected concerns and the arrows the analysis or transformation rules. The analysis rules are defined by labeled arrows starting with CR, the transformation rules are defined by labeled arrows starting with CT.

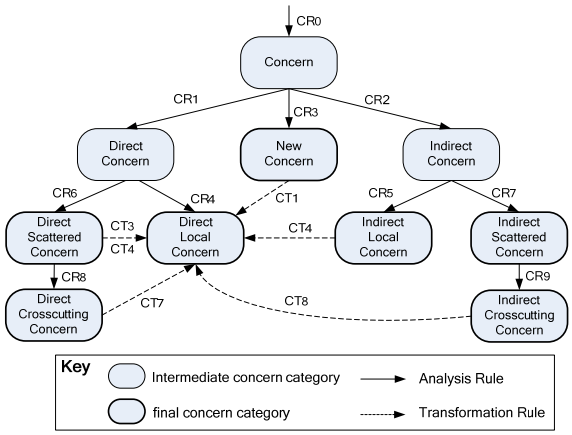

Figure 3. Transition diagram for characterization of concerns

During the analysis each concern is eventually characterized in one of the following type of concerns: *New Concern,* is a new concern that has not been considered in the architecture. *Direct Local Concern,* is a concern that can be directly addressed by one module in the architecture. *Indirect Local Concern* is a concern that is not yet realized in the architecture but can be realized in one module. *Direct Scattered Concern* is a concern that is realized by the architecture but is scattered over multiple modules. *Indirect Scattered Concern* is a concern that is not realized by the architecture yet but will be scattered over multiple modules if so. *Direct Crosscutting Concern* is a scattered over several modules of which at least one is tangled (see characterization modules). *Indirect Crosscutting Concern* will be scattered over several modules of which at least one is tangled (see characterization modules).

The heuristic rules for characterizing concerns are given in Table 2. The left column defines the possible patterns in the DMM. Each pattern is related with a corresponding heuristic rule. Each rule is defined in the following format:

```
IF <condition> THEN <consequent>
```

Whenever the pattern is found in the DMM and the condition of an analysis rule is met, the concern is categorized into a specific category. Note that rules CR0, CR1 and CR2 represent the rules for initializing the DMM as defined in Table 1. These rules provide the characters "D" and "I" in the DMM to denote whether the concern is direct or indirect. After the initialization each concern is further automatically characterized by analyzing the DMM.

Table 2. Patterns and heuristics for characterizing concerns in the DMM

DMM Pattern	Category	Heuristic Rule
	D: Direct Concern I: Indirect Concern	**CR0.** SELECT CONCERN from Preparation Phase **CR1.** IF CONCERN is directly addressed by a module THEN CONCERN is a DIRECT CONCERN **CR2.** IF CONCERN is indirectly addressed by a module THEN CONCERN is an INDIRECT CONCERN
M1 M1 M3 C	N: New Concern	**CR3.** IF CONCERN is not addressed by any module THEN CONCERN is a NEW CONCERN
M1 M1 M3 C D	DL: Direct Local Concern	**CR4.** IF DIRECT CONCERN is addressed by a single module THEN DIRECT CONCERN is a DIRECT LOCAL CONCERN
M1 M1 M3 C I	IL: Indirect Local Concern	**CR5.** IF INDIRECT CONCERN is addressed by a single module THEN INDIRECT CONCERN is an INDIRECT LOCAL CONCERN
M1 M1 M3 C D D D	DS: Direct Scattered Concern	**CR6.** IF DIRECT CONCERN is addressed by multiple modules THEN DIRECT CONCERN is a DIRECT SCATTERED CONCERN
M1 M1 M3 C I I I	IS: Indirect Scattered Concern	**CR7.** IF INDIRECT CONCERN is addressed by multiple modules THEN INDIRECT CONCERN is a INDIRECT SCATTERED CONCERN
M1 M1 M3 C1 D D D C2 D/I	DX: Direct Crosscutting Concern	**CR8.** IF DIRECT SCATTERED CONCERN is addressed by at least one module THEN DIRECT SCATTERED CONCERN is a DIRECT CROSSCUTTING CONCERN
M1 M1 M3 C1 I I I C2 D/I	IX: Indirect Crosscutting Concern	**CR9.** IF INDIRECT SCATTERED CONCERN is addressed by at least one module THEN INDIRECT SCATTERED CONCERN is a INDIRECT CROSSCUTTING CONCERN

In addition to rules for characterizing concerns we have also defined a set of rules for characterizing modules as defined in Table 3. Modules are characterized as follows: *Direct Cohesive Module* addresses only one concern directly. *Indirect Cohesive Module* addresses only one concern indirectly. *Direct Tangled Module* addresses multiple concerns directly. *Indirect Tangled Module* addresses multiple concerns indirectly. The characterizations are defined through applying a set of heuristic rules. Here we also distinguish between analysis rules and transformation rules. After the analysis of the concerns we characterize the modules again by checking the related DMM pattern and the conditions specified in the rules. Figure 4 depicts the transition diagram for characterizing modules.

Table 3 Patterns and heuristic rules for characterizing modules in the DMM

DMM Pattern	Category	Heuristic Rule
	D: Direct Module I: Indirect Module	**MR0.** SELECT Module from Preparation Phase **MR1.** IF MODULE addresses a concern directly THEN MODULE is a DIRECT MODULE **MR2.** IF MODULE addresses a concern indirectly THEN MODULE is an INDIRECT MODULE
M1 M1 M3 C1 D C2	DC: Direct Cohesive Module	**MR3.** IF DIRECT MODULE addresses a single concern THEN DIRECT MODULE is a DIRECT COHESIVE MODULE
M1 M1 M3 C1 I C2	IC: Indirect Cohesive Module	**MR4.** IF INDIRECT MODULE addresses a single concern THEN INDIRECT MODULE is an INDIRECT COHESIVE MODULE
M1 M1 M3 C1 D C2 D	DT: Direct Tangled Module	**MR5.** IF DIRECT MODULE addresses multiple concerns THEN DIRECT MODULE is a DIRECT TANGLED MODULE
M1 M1 M3 C1 I C2 I	IT: Indirect Tangled Module	**MR6.** IF INDIRECT MODULE addresses multiple concerns THEN INDIRECT MODULE is an INDIRECT TANGLED MODULE

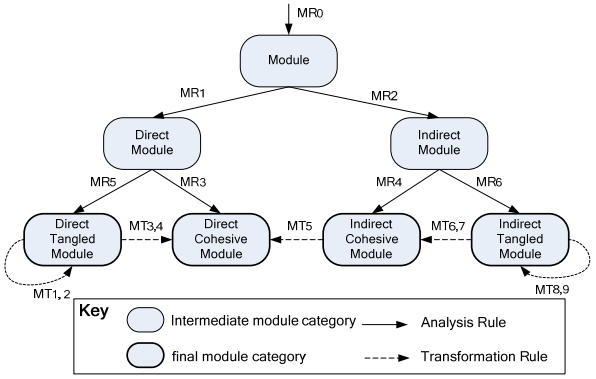

Figure 4.Transition diagram for characterization of modules

2.2.3 Measure Impact of Concerns

In the previous activity we have characterized the mapping between the concerns and modules of the window manager software architecture. In this activity we measure the impact of concerns and tangling of modules, based on the mappings in the concern-module DMM. For this we use a set of metrics for measuring the scattering degree of concerns and tangling degree in modules.

2.3 Transformation Phase

The analysis depicts how concerns are mapped to architectural modules and provides a clear insight in the scattering of concerns and tangling of modules. In the transformation phase the output of the analysis phase is used to enhance the modularity of the architecture. For this the architecture is first modeled using DSM. To enhance the modularity of the architecture, in COSAAM we aim to increase the number of *Direct Local Concerns* and the number of *Direct Cohesive Modules* in the Concerns-Modules DMM. For this a set of transformation rules (not depicted here) are applied to reduce scattering and tangling. The global process for this is shown in Figure 5.

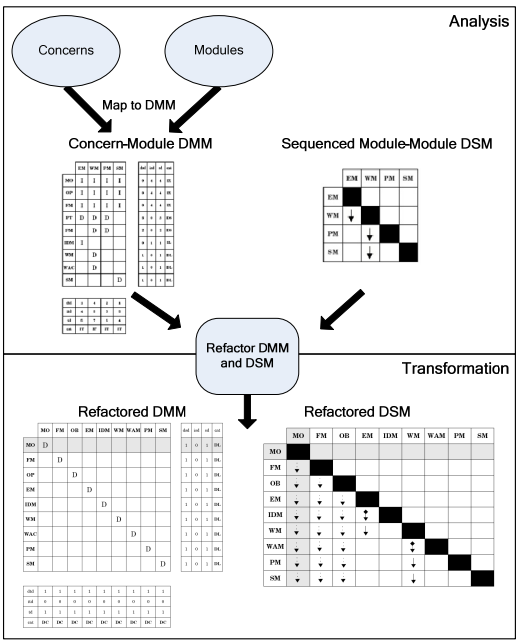

Figure 5. Refactoring DMM

The *Concern-Module DMM* is manipulated to refactor the allocation of concerns to modules. In essence the following primitive actions can be taken in the concern-module DMM: *add concern*, *remove concern, add module, remove module, change mapping relation (direct or indirect).* The manipulations on DMM will have also an impact on DSM for the architecture. Here we can in principle utilize the following actions: *add module, remove module, add relationship among modules, remove relationship among modules, change relationship.*

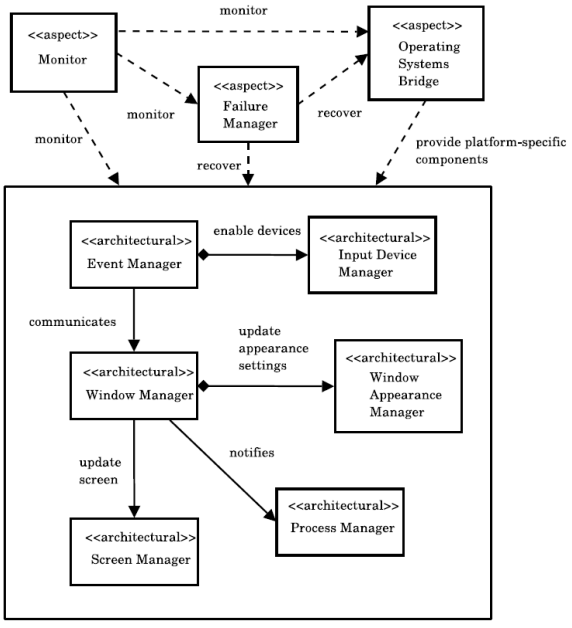

Figure 6. WMS Architecture after analysis and refactoring

Due to size restrictions it is not possible to demonstrate the set of transformations. A detailed description of the evolution of the

window manager architecture is provided in [11]. The final version of the transformed window manager architecture, after nine iterations, is shown in Figure 6.

3. RELATED WORK

The development of complex software systems carries a large initial investment and a considerable risk. Therefore have various architecture analysis methods been proposed in the past years that are used to analyze whether software architectures meet certain quality requirements. An extensive survey of these methods can be found, for example, in [2].

The application of design structure matrices to software architecture design is a relatively new research area. Its application to general product development processes is quite well understood and discussed, for example, in [8][10]. Besides managing dependencies in software architectures DSM has also been applied for analyzing the modularity of aspect-oriented designs [5][6]. Lopes and Bajracharya have demonstrated that aspects can make software architecture designs more valuable. However, in [7] they show that aspects can provide a negative contribution to the value of a software architecture design. By applying the Net Option Value and DSM techniques to a set of conventional supports and aspect oriented software architecture designs they conclude that aspects should avoid the introduction of additional dependencies and hide design parameters from other modules in order to be of value for the design. In this work, the authors also introduce preliminary design guidelines for aspects that aim at minimizing the dependencies between modules independent of their aspectual or non-aspectual nature. The application of the design guidelines is based on a DMM based classification of modules according to their dependency relationships. This differs from the net option value based approach, because it considers purely structural properties, like cohesiveness, crosscutting and tangling of modules in the software architecture design. The application of the net option value requires the estimation of direct and derived cost measures, as well as an estimation of the return on investment that is to be expected from the reuse of the module.

4. CONCLUSION

Current scenario-based software architecture analysis methods aim to analyze the impact of stakeholder concerns on the architecture. Unfortunately the notion of concern is not a first class abstraction in these approaches and the analysis is primarily based on the impact of scenarios. This leads to a partial understanding of the impact of concerns and as such provides risks for the optimal refactoring of the architecture.

We have introduced the Concern-Oriented Software Architecture Analysis Method that explicitly uses the notion of concern in the analysis. The key tools in the approach are Dependency Matrices that depict the dependencies of module elements. Our study shows that an explicit notion of concern in the analysis facilitates the understanding on the impact and as such provides better support for maintenance. For example, in our analysis an important conclusion was that the introduction of architectural aspects is a trade-off among cohesion and coupling. On the one hand aspects support cohesion of the architectural modules, on the other hand additional couplings with the modules. This was obvious in the dependency matrices, the number of mappings in the concern-architecture DMM are reduced while the couplings in the architecture DSM increase. In our future work we will develop a tool that implements the COSAAM process. This will enable us to experiment and validate COSAAM for a broader set of applications.

ACKNOWLEDGEMENTS

This work is supported by European Commission Grant IST-2-004349: European Network of Excellence on AOSD (AOSDEurope) and the *Aspect-Oriented Software Architecture Design project* which is funded by the Dutch Scientific Organisation in the *Jacquard Software Engineering Program*.

REFERENCES

[1] Bass, L., Clements,P., and Kazman, R. Software Architecture in Practice, second edition, Addison-Wesley 1998.

[2] Dobrica, L. and Niemela, E. A survey on software architecture analysis methods. IEEE Trans. on Software Engineering, Vol. 28, No. 7, pp.638-654, July 2002.

[3] Lattix Inc. http://www.lattix.com.

[4] Sangal, N., Jordan, E., Sinha,V., and Jackson, D.. Using Dependency Models to Manage Complex Software Architecture. In OOPSLA ’05:, pages 167–176, New York, NY, USA, 2005.

[5] Sullivan, K. J., Griswold, W.G., Cai, Y. and Hallen, B. The Structure and Value of Modularity in Software Design. SIGSOFT Softw. Eng. Notes, 26(5):99–108, 2001.

[6] Videira Lopes, C. and Bajracharya, S. An Analysis of Modularity in Aspect Oriented Design. In AOSD '05 Proceedings, pages 15-26, ACM Press, 2005.

[7] Videira Lopes, C. and Bajracharya, S. Assessing Aspect Modularizations Using Design Structure Matrix and Net Option Value. In Trans. Aspect-Oriented Software Development I, pages 1–35, 2006.

[8] Yassine, A. An Introduction to Modeling and Analyzing Complex Product Development Processes Using the Design Structure Matrix (DSM) Method. 2002.

[9] Tekinerdogan, B. ASAAM: Aspectual Sofware Architecture Analysis Method. In Working IEEE/IFIP Conference on Software Architecture, pp. 5-14, 2004.

[10] Danilovic, M. and Sandkull, B. The use of dependency structure matrix and domain mapping matrix in managing uncertainty in multiple project situations. *International Journal on Project Management, 3:193-203, 2005*

[11] Scholten, F. The Concern-Oriented Software Architecture Analysis Method. MSc. Thesis. University of Twente, 2007.