Using Variability Modeling Principles to Capture Architectural Knowledge

Marco Sinnema
University of Groningen
PO Box 800
9700 AV Groningen
The Netherlands
+31503637125
m.sinnema@rug.nl

Jan Salvador van der Ven
University of Groningen
PO Box 800
9700 AV Groningen
The Netherlands
+31503637125
j.s.van.der.ven@rug.nl

Sybren Deelstra
University of Groningen
PO Box 800
9700 AV Groningen
The Netherlands
+31503637125
s.k.deelstra@rug.nl

ABSTRACT
In the field of software architectures, there is an emerging awareness of the importance of architectural decisions. In this view, the architecting process is explained as a decision process, while the design and eventually the software system are seen as the result of this decision process. However, the effects of different alternatives on the quality of the system often remain implicit. In the field of software product families, the same issues arise when configuring products. We propose to use the proven expertise from COVAMOF, a framework for managing variability, to solve the issues that arise when relating quality attributes to architectural decisions.

Keywords
Architectural Decisions, Architectural Knowledge, Quality Attributes.

1. Introduction
Software architecture documents contain only a subset of the architectural knowledge that is used to create an architecture. In particular during the development of large software systems, a lot of knowledge stays implicit. The effects of different alternatives on the quality attributes of a system, or the rationale of decisions are typical examples of tacit architectural knowledge that is slowly lost, because it is not recorded and people tend to forget it.

The knowledge about the effects on the quality of the system is very important because of two reasons. First, this knowledge increases the understandability of the decision process, as it shows why certain alternatives were not viable, for example. Second, the knowledge has to be available when there is a need to change the system later on, e.g. changing the decision made. It is therefore evident to write these relationships down, preferably in formalized form.

The effects of architectural decisions on the functional and quality aspects of a system are often hard to make explicit, however. First, this is because architectural decisions lack a first class representation in the architecting process [3]. Second, the effects are often imprecise, as they result from predictions that are based on the experience of software architects, from previous projects, or from pilot projects. Usually, some hints about the effects of the decisions on the quality exist, but these are hard to formalize. Finally, the decision process is a trade-off process between quality attributes. The effects of decisions tend to become very complex, e.g. if many decisions affect the same quality attribute.

In the field of software product families, similar problems are faced for modeling variability. To derive a product from a product family, engineers have to decide which alternatives they should choose from the reusable asset base. Variability modeling is concerned with modeling how different alternatives in the product family affect the functional and quality aspects of a product.

The COVAMOF variability modeling framework was developed to be able to deal with the imprecise and incomplete nature of the effect of decisions on quality attributes. We think the experiences that are gained from developing and applying this framework could help us in gaining more insight in how to model the effects of the architectural decisions on the quality of the system. In this paper, we show how the concepts and issues of architectural knowledge map to concepts and issues that are addressed by COVAMOF.

To this purpose, this paper is organized as follows. In the following section, the concept of architectural knowledge is explained, based on some recent work from the research community. Section 3 explains the COVAMOF concept, and explains how quality attributes can be modeled in software product families. Then, the two approaches are compared, and the corresponding concepts are highlighted. Section 5 presents our vision for the future, and Section 6 concludes this paper.

2. Architectural Knowledge
There is no agreement about what architectural knowledge embodies. Various models of architectural knowledge and the relationships have been proposed [1][11][12], for example. However, many approaches refer to similar architectural notions,

SHARK '06, June 11, 2006, Torino, Italy.

albeit that they use different terms to refer to them. Most emphasize the architectural choices (also referred to as design decisions), as essential architectural knowledge [3][6], for example. The architectural choices consist of considered alternatives, rationale, assumptions, etc. [11][12] and directly influence the design [5]. In this section we describe the concepts that are commonly used in research on architectural choices, and show how different approaches refer to them.

2.1 Architectural Choices and Alternatives

In a decision process *alternatives* are evaluated. These alternatives are possible solutions in the given situation. De Boer et al. [1] and [12] van der Ven et al, describe how these alternatives can be represented as first-class entities called alternatives. Tyree and Akerman [11] describe these alternatives as positions. Kruchten et al. [6] identify the 'Is an Alternative to' relationship as a possibility to express alternatives of a decision.

An architectural *choice* is the selection of one of the alternatives. Kruchten et al. represent this choice textually as a part of the rationale. Tyree and Akerman consider choice as the selection of one of the positions, similar to van der Ven et al., who use the choice to select an alternative. In the work of de Boer et al. the decision element is used to represent the choice for an alternative.

2.2 Issue

In a decision process, *issues* arise on which decisions have to be made. In other words, what do we need to solve by creating alternatives and selecting one? These issues (or problems) are the direct result of the requirements for the system [12]. De Boer et al. call the issue a decision topic. Kruchten et al. do not specify a problem element.

2.3 Decisions and Rationale

In all the work about architectural knowledge, *rationale* is described as essential to understanding the choices. It is often referred to as the 'why' (Kruchten et al), the 'reason' (van der Ven et al.), or 'argument' (Tyree and Akerman) of the decision. De Boer et al. deliberately do not include the rationale as an element, because they believe it is scattered across the model and cannot be captured in a single element.

2.4 Quality Attributes and Trade-offs

Architectural decisions are typically strongly related to each other. Kruchten et al. describe several of these relationships between architectural decisions. One of those types of relationships is interesting in the context of this paper: the influence the architectural decisions have on the quality of the system. Kruchten et al. refer to these types of relationships as *categories*. Van der Ven et al. relate architectural decisions to the functional and non-functional requirements. In the work of de Boer et al. the quality attributes can be represented by the element *concern*. Tyree and Akerman describe the possibility to add related non-functional requirements to a decision. In all of these cases, this relationship between architectural decisions remains implicit, or is just represented as text. Support for helping the architect in the trade-off process, where the effects of decisions on quality attributes is used to guide the decision process, is not found in any of the approaches.

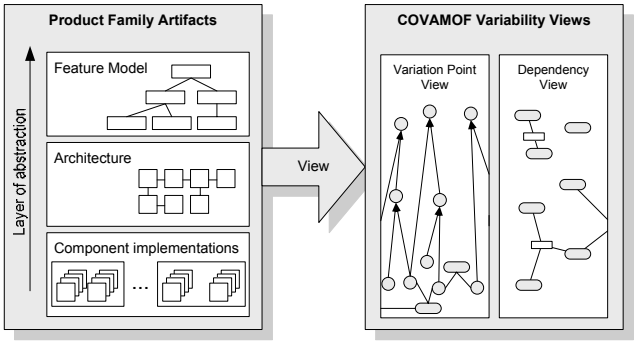

Figure 1 COVAMOF provides several views on the variability provided by a software product family

3. COVAMOF

The problems with incomplete and imprecise knowledge do not only occur in the context of architectural knowledge, but have also been identified in the field of software product families in Sinnema et al. [10]. On the one hand this work provides more detailed insight in the problems with tacit, incomplete and imprecise knowledge and illustrates these problems with examples from industrial case studies. On the other hand it explains how the COVAMOF framework [9] addresses these issues in the context of software product families. COVAMOF is a variability modeling framework that consists of models, tooling and processes that support engineers in the development of product families as well as the configuration of individual products from a product family.

In this section, we briefly introduce the concept of software product families and show how COVAMOF deals with incomplete and imprecise knowledge and quality attributes. In the next section, we show how the concepts and issues addressed by COVAMOF map to capturing architectural knowledge.

3.1 Software Product Families

The basic philosophy of software product families is intra-organizational reuse through the explicitly planned exploitation of similarities between related products. This philosophy is realized by maintaining a reusable asset base, where assets incorporate variability to deal with product differences. On the one hand, this philosophy has been adopted by a wide variety of organizations and has proved to substantially decrease costs and time-to-market, and increase the quality of their software products [7]. On the other hand, case studies at industrial software product families indicated that still some challenges exist that are primarily related to complexity and incomplete and imprecise knowledge [2].

3.2 Variation Points, Variants, and Dependencies

In response to these problems, the COVAMOF variability modeling framework has been developed. The idea behind COVAMOF is that it provides several views on the variability that is provided by the product family artifacts (see also [9] and Figure 1). These views are based on the knowledge in the associated variability model. The main entities in this model are Variation Points and Dependencies.

Variation point entities in COVAMOF represent the locations at which a choice is provided by the product family and *variant* entities represent the options that are available at these variation

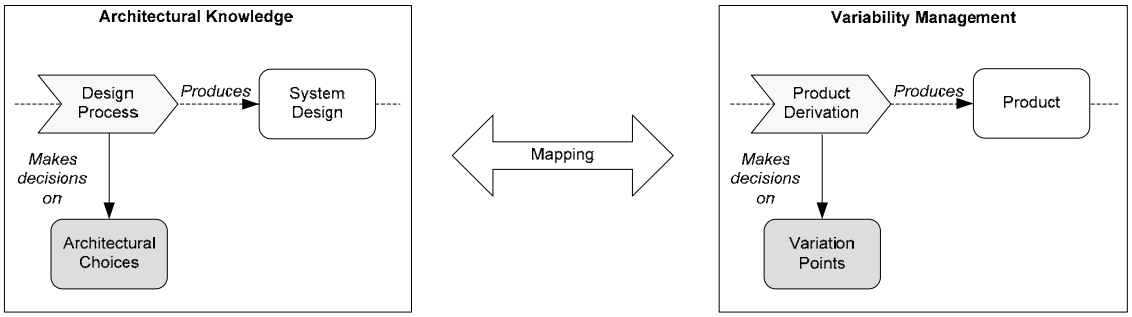

Figure 2 The correspondence between Architectural Knowledge and Product Family Engineering. Where the design process in architectural development breaks down into making decisions on architectural choices and producing a system design, product derivation is basically the process of making decisions (referred to as binding) on variation points to end up with a software product.

points. A variation point entity has a number of properties, for example the moment in the lifecycle at which the choice is bound (binding time), and storing the reason why a choice is provided (rationale).

The *dependencies* specify a mapping from the configuration of a set of variation points to a value in a specific 1-dimensional domain. This means that if you select variants at a variation point (called the configuration), the dependency maps this selection to a value for some system property of the final product.

3.3 Dependencies and Quality Attributes

Examples of those system properties are the consistency of the selected variants, the memory consumption or performance of the final product, or other quality attributes. If we focus on quality attributes in general, COVAMOF refers to the set of variation points that influence the quality attribute of the dependency as the *associated* variation points. To accommodate for incompleteness and impreciseness, it distinguishes between three different types of associations, i.e. abstract, directional and logical associations.

Abstract: The only information product family experts have on abstractly associated variation points is that its configuration influences the value of the quality attribute related to the dependency. There is no information available on how a (re)binding will effect this value.

Directional: Directional associations do not only specify that its configuration influences the quality attribute related to the dependency, but it also describes some information on how the value depends on its binding. The effect of a (re)binding of a directionally associated variation point is not necessarily fully known and documented, and is not specified in a formal manner. The dependency entity, however, describes this effect as far as it is known to the experts.

Logical: Dependencies furthermore contain logically associated variation points. The effect on the quality attribute of the dependency is fully known and is specified as formal knowledge.

As dependencies can have combinations of different associations, COVAMOF allows engineers to specify the influence of variants by a combination of incomplete, imprecise knowledge. In [10], we give a more detailed description of these association types and illustrate them with examples from an industrial case study.

3.4 Interaction between Quality Attributes

In industrial product families, engineers have to consider large numbers of dependencies (such as quality attributes) during the derivation process, each one associated with a number of variation points ranging from one to almost all variation points. Therefore, a lot of variation points are typically associated to more than one dependency. As a result, configuring a variation point in the context of the optimization of one dependency may influence the value of all other dependencies that share that variation point. We refer to this mutual influence between dependencies as *dependency interaction*. As the optimal values of those dependencies are often contradicting, the optimization of dependencies and their related quality attributes is often a complicated task and involves an extensive amount expert knowledge.

Although the sets of dependencies that interact can easily be generated from the dependency entities and their associations, the COVAMOF variability model also explicitly captures dependency interaction entities in the variability model. They specify, for a set of dependencies, how to cope with the shared associated variation points during product derivation. This textual specification is documented by product family experts and contains a strategy for developing a reasonable first guess during the initial phase, a strategy to optimize the values in the iteration phase, as well as guidance for making trade-offs between interacting dependencies (in particular trade-offs between quality attributes).

3.5 Experimental Results

The applicability of COVAMOF has been tested by experiments in several industrial software product families. It appeared that the decisions made by engineers during product derivation had improved and that they were able to make better estimations for the variants and values that have to be selected. This reduced the time and effort required to derive a typical product from the product family by at least 90% [8].

4. COVAMOF and Architectural Decisions

Whereas in the previous section we showed how COVAMOF supports the engineers in a software product family context, in this section we explain how the concepts of COVAMOF can be used to support architects in the architectural knowledge field. To

Table 1. The mapping of architectural concepts onto concepts in the COVAMOF framework.

Architectural Concept	COVAM OF Concept
Design	Product
Architectural Choice	Variation Point
Alternative	Variant
Issue	Rationale of Variation Point
Decision	Decision
Rationale	*none*
Quality Attribute	Dependency
Trade-off	Dependency Interaction

this purpose, we explain how the concepts in the architectural knowledge field from section 2 map onto concepts in the COVAMOF variability model. This mapping is based on the correspondence between architectural development and product family engineering as visualized in Figure 2. The following subsections correspond to the subsections of section 2. In the last subsection we show what benefits this mapping brings to the architects that work with architectural knowledge.

4.1 Architectural Choices and Alternatives

Section 2.1 explains how the knowledge about the choices and their alternatives for a design could explicitly be represented. As variation points and variants are first class entities in COVAMOF and describe a similar mechanism, they can represent these choices and alternatives, respectively. In this respect, these choices are more or less seen as variation points with a binding time of "development" for a single product.

4.2 Issue

In COVAMOF, the "rationale" attribute of a variation point describes the reason why the product family provides choices between the associated variants. Similarly, section 2.2 proposes to represent the reason for a certain architectural choice and its alternatives to exist, also referred to as the "issue". This issue aspect therefore maps directly onto the "rationale" attribute of variation point.

4.3 Decisions and Rationale

In COVAMOF, products are modeled as a set of decisions, where decisions refer to the actual binding of variants to a variation point in a specific product. Although this concept of decision in COVAMOF corresponds to the concept of decision in the architectural knowledge field, i.e. the alternative that is realized in the final product, decisions in COVAMOF do not accommodate for the specification of the rationale behind the decision. While tailoring the concepts of COVAMOF for the application in an architectural knowledge context, decision entities should therefore be extended with an additional "rationale" property.

Note that the way COVAMOF represents products (i.e. as a set of decisions) corresponds to the view on architectural choices *being* the final architectural design as viewed by Jansen and Bosch [5].

4.4 Quality Attributes and Trade-offs

In COVAMOF, dependencies are used to specify the influence of a certain set of decisions on the actual value of the relevant quality attributes (see also section 3.3). Architects can specify this influence by formal or documented knowledge or a combination of both. Architects can formally specify that certain variation points influence a dependency for example, and document hints, reference data, and specify directions in which quality attributes change under reselection. They therefore allow for using incomplete as well as imprecise knowledge.

As architectural choices are represented as variations points and their alternatives are represented by variants, the dependencies of COVAMOF can easily be used to represent the influence of the alternatives on the quality attributes. Moreover, in this way the dependency interactions can be used to assist the experts in dealing with the complex interactions between different alternatives and making trade-offs between several different quality attributes.

4.5 Summary

In the previous subsections we showed how the concepts of COVAMOF can be used to capture the architectural knowledge that is required to support software architects in designing a software system. This mapping is summarized in Table 1 and visualized in Figure 3. The table maps each architectural concept (left column) onto a concept in the COVAMOF variability modeling framework (right column).

This mapping makes it possible to use the COVAMOF tool suite and method for modeling architectural knowledge. Figure 3 shows the COVAMOF concepts, their graphical representation, together with the corresponding concepts from architectural knowledge domain. Notice that the decisions at architectural level affect one design (e.g. the system that is being designed), while with product families the decisions are made per product.

By using the approach adopted by COVAMOF for capturing architectural knowledge, it is possible to explicitly deal with the implications attached to tacit, documented and formalized architectural knowledge, as it does not require a complete and fully formalized model in order to be useful during architecture

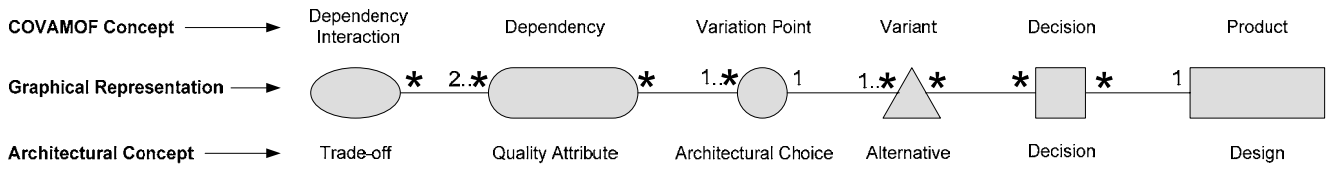

Figure 3 Graphical representation of the mapping of Architectural Concepts onto Architectural Concepts.

design. This allows organizations to start with a minimal amount of formalization that can pay off immediately. The benefit provided by a COVAMOF model is that it can be gradually extended when more precise knowledge becomes available, or when more benefits are perceived for the externalization.

Second, the way in which dependencies are modeled in COVAMOF enables partially formalizing the quality of the system, and incorporating (links to) documented and tacit knowledge. It provides a central and structured repository for obtaining all architectural knowledge. This reduces the gaps between tacit, documented and formalized knowledge, and thus the problem of engineers not being able to find all relevant knowledge.

5. Vision

In the previous sections we explained how we can use variability modeling principles to capture architectural knowledge. In this section, we show how we can integrate these two worlds into one solution for developing software.

This vision is based on the current trend in product families that the moment of binding is shifting more and more to later moments in the product life cycle. Whereas in the early 90's, most variation points were bound at pre-compile time, nowadays this focus has moved to run-time configuration. We expect this trend not only in the field of product families, but also in the development of single systems, which is currently the current focus of architectural knowledge research. As a result, the focus of the Design Process (depicted in the top-left of Figure 4) will shift from producing a single system design, to the development of a design that accommodates for variability. This variability encompasses both the pre-runtime variation points (i.e. to vary functionality between products for different customers) and runtime variation points (i.e. the choices individual customers can make in their product).

To explain this integrated approach, we have to take a closer look at the correspondence between Architectural Knowledge and Variability Management and the concept of binding time for variation points. The *binding time* of a variation point refers to the latest moment in the development life-cycle where engineers have to make the decision on which variants to use in a product. In current variability management practice, illustrated in the top-right Figure 4, we distinguish (amongst others) between the *pre-runtime* and *runtime* binding times. In this respect, we can not just use the concepts of COVAMOF to model architectural choices (as explained in section 4), but also view architectural choices as being variation points with a binding time at the development of the design of a product with variability. This product with variability can be seen as the design of a product family from which individual products with run-time variability can be derived.

This vision is visualized in the bottom part of Figure 4. It shows that variation points occur at three different binding times. First, engineers make decisions on the variation points for which no choices will be available in the product family. Second, the resulting product family design is used to derive individual products from the product family. In this product derivation process, engineers make decisions on variation points that realize the customer needs. They leave the variation points that are not bound open for customers to configure their individual product at runtime. Note that, in accordance to how pre-runtime variation

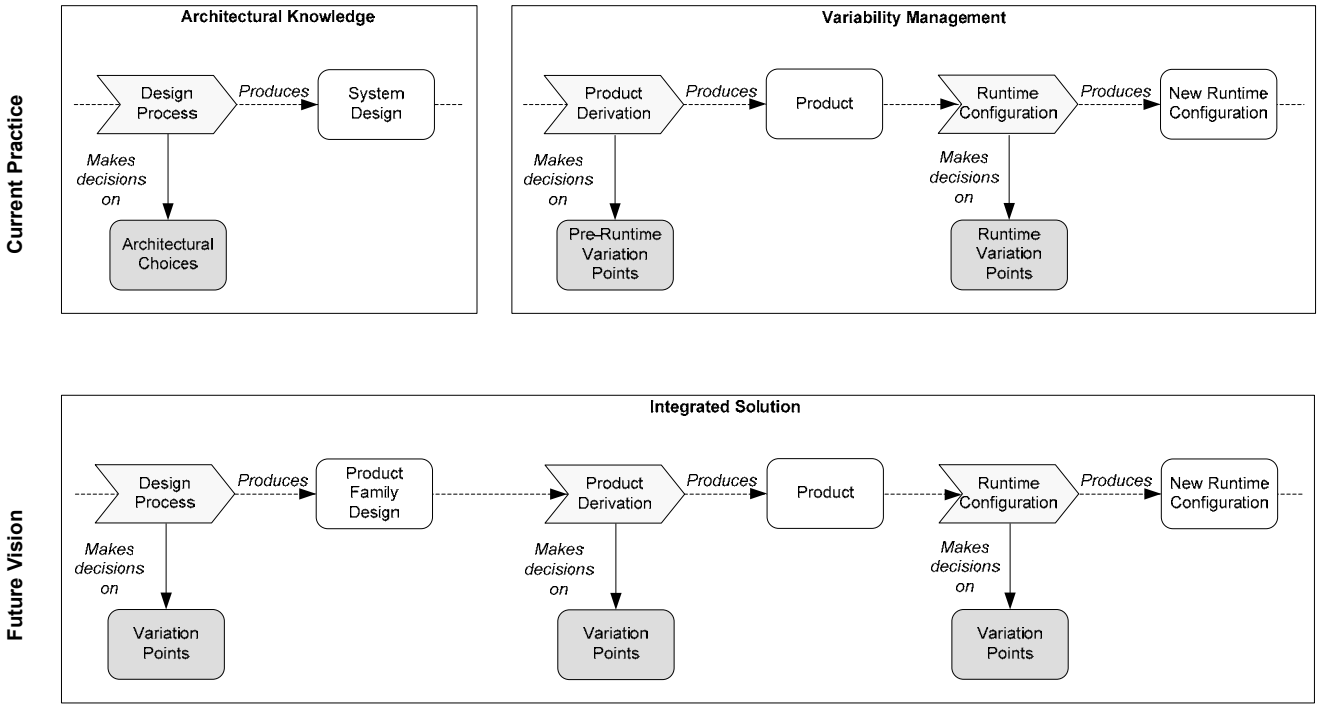

Figure 4 Vision on future software development. A uniform framework is used to support the engineers in all stages of developing software, where the Design Process can be seen as configuring variation points (i.e. architectural choices) with a binding time "development".

points can specify the variation points available at run-time, architectural choices can specify the pre-runtime variation points available in the product family during product derivation.

In the integrated solution of Figure 4, we refer to the architectural choices, pre-runtime variation points and runtime variation points as just variation points to show we can model them in a uniform way. As a result, engineers can use the modeling concepts and support (e.g. processes and tools) applied in product family engineering nowadays to support the development of products (possibly with variability).

6. Conclusion

Although it seems that the worlds of architectural knowledge and variability modeling are very different, the problems they face have a common structure. We have shown that solutions that are already used in the field of software product families can help solving the problems faced in the architectural knowledge field.

The idea behind COVAMOF is to enable tool support to manage complexity and reduce the dependency of organizations on experts. While a first benefit externalization lies in automating management of simple in- and exclude relations that can easily be formalized, a primary benefit of externalization is that it alleviates the vulnerability to knowledge starvation, i.e. loss of important knowledge when experts leave the organization. An second important benefit of the externalization principles of COVAMOF is being able to handle documented knowledge, which enables less experienced engineers to make estimates for quality attributes.

We envision that creating software will use techniques similar to the ones presented in this paper for the whole software design cycle. The actual taking of decisions will slowly move from architecture time, to development time and eventually to runtime.

7. References

[1] R. C. de Boer, R. Farenhorst, V. Clerc, J. S. van der Ven, P. Lago, H. van Vliet, A Model for structuring Software Architecture Project Memories, Proceedings of the 8th International Workshop on Learning Software Organizations, 2006.

[2] J. Bosch, Design and use of software architectures: adopting and evolving a product line approach, Pearson Education Addison-Wesley and ACM Press), ISBN 0-201-67494-7, 2000.

[3] J. Bosch, Software architecture: The next step, Proceedings of the First European Workshop on Software Architecture (EWSA), Volume 3047 of LNCS, Springer, pp. 194-199, 2004.

[4] S. Deelstra, M. Sinnema, J. Bosch, Product Derivation in Software Product Families; A Case Study, Journal of Systems and Software, Vol 74/2 pp. 173-194, January 2005.

[5] A.G.J. Jansen, J., Bosch, Software architecture as a set of architectural design decisions, Proceedings of the 5th IEEE/IFIP Working Conference on Software Architecture (WICSA 5), 2005.

[6] P. Kruchten, P. Lago, H. van Vliet, Building up and exploiting architectural knowledge, submitted to the Second International Conference on the Quality of Software Architectures (QoSA 2006), 2006.

[7] Linden, F. van der, 2002. Software Product Families in Europe: The Esaps & Cafe Projects. IEEE Software 19 (4), 41–49.

[8] M. Sinnema, S. Deelstra, P. Hoekstra, The COVAMOF Derivation Process, Proceedings of the 9th International Conference on Software Reuse (ICSR 2006), June 2006.

[9] M. Sinnema, S. Deelstra, J. Nijhuis, J. Bosch, COVAMOF: A Framework for Modeling Variability in Software Product Families, Proceedings of the Third Software Product Line Conference (SPLC 2004), Springer Verlag Lecture Notes on Computer Science Vol. 3154 (LNCS 3154), pp. 197-213, August 2004.

[10] M. Sinnema, S. Deelstra, J. Nijhuis, J. Bosch, Modeling Dependencies in Product Families with COVAMOF, Proceedings of the 13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2006), March 2006.

[11] J. Tyree and A. Akerman, Architecture decisions: Demystifying architecture. IEEE Software, 22(2):19–27, 2005.

[12] J. S. van der Ven, A. G. J. Jansen, J. A. G. Nijhuis, J. Bosch, Design decisions: The bridge between rationale and architecture, In A. H. Dutoit, R. McCall, I. Mistrik, and B. Paech (Editors), Rationale Management in Software Engineering, Chapter 16, Springer-Verlag, March 2006.