Evaluation of Product Line Architecture Design Methods

Mari Matinlassi

VTT Electronics, P. O. Box 1100, FIN-90571 Oulu, Finland
E-mail: Mari.Matinlassi@vtt.fi

Abstract. The role of software architecture has changed. Product line architecture has also become a recently emerged discipline. A number of software architecture design methods have been developed but only three of them are known to answer the needs of software product lines. These methods are SPLIT, CoPAM and FORM. In this paper, an evaluation framework is introduced for comparing product line architecture design methods. The selected design methods for product line architectures are evaluated according to the framework and the results are described in the form of a comparison table.

Introduction

The design and documentation of software architecture [5] has become more important. Architectural views have been the basis for a number of design methods developed and used during last few years [4], [6], [9], [10], [17]. Despite the fact that these methods are capable and exhaustive in their own way; none of them concerns the product line approach to the architectural design deeply enough. However, as far as is known, there are only three methods answering the needs of product lines: SPLIT [7], CoPAM [2], [3] and FORM [14].

This paper evaluates three selected design methods for product line architectures and describes the results in the form of a comparison table. The aim is to understand the current state of product line architecture design methods and to find out the differences and similarities between these methods.

Evaluation Framework

An evaluation framework that is introduced in Table 1 is used as an analysis tool. The framework is based on three sources. The first is the NIMSAD (Normative Information Model-based Systems Analysis and Design) evaluation framework [11]. In addition to the NIMSAD framework, the definition of a method and its ingredients [16] has influenced the third element of the framework, i.e. the method contents. The third source for the evaluation framework is the evaluation framework for component-based software development methodologies [8]. The goal of the evaluation is not to rate the methods but to find out if the elements defined in the framework are consid-

ered by the method. If and when the elements are considered, the objective is to investigate how it has been done.

Table 1. The elements of the framework and the questions used in the analysis.

Elements	Questions
Method Context	
Specific goal	What is the specific goal of the method?
Method inputs	What is the starting point for the method?
Method outputs	What are the results of the method?
Method User	
Motivation	What are the *user's* benefits when using the method?
Needed skills	What skills does the user need to accomplish the tasks required by the method?
Method Contents	
Models	What are the models represented and manipulated by the method?
Method structure	What are the design steps that are used to accomplish the method's specific goal?
Tool support	What are the tools supporting the method?
Method Validation	
Maturity	Has the method been validated in practical industrial cases?
Domains	What are the domains the method is validated in?

Evaluation

the evaluation results are divided into four elements: *context, user, contents and validation*. Method *context* means the environment where the method is going to be used. As seen in Table 2, the starting point for all the methods is the same: requirements specifications. Requirements are taken as inputs to each method, but after product line architecture design, the method outputs differ. FORM seems to produce the most in-depth method outputs by generating results that are quite close to the implementation, whereas CoPAM takes a wider insight into the issue by also considering the business and organizational aspects. The results of the SPLIT method are at the domain modeling level, resulting in a definition of a product line by means of requirements, architecture, components and variation points.

A method is engineered to solve a problem or problems. It is obvious that all methods, as well as software development methods, must have a method *user*. In the context of the product line software design method, the method users are the product line architects. Table 2 also summarises the motives for architects to use the three methods and skills that are needed to get full benefit from the methods. A surprising fact is that none of the methods under evaluation highlight the user benefits of the methods'. The use of every method requires some skills. SPLIT and CoPAM have recognized the fact that a successful method utilizes skills that are already widely known among software professionals. The FORM method differs with the others in at least in two ways: first, it defines notation, techniques and a tool, ASADAL [1], explicitly. Second, the skills that are explicitly specified are not widely known among software developers. Under the interpretation above, SPLIT and CoPAM are more practical

methods and aimed at industrial use, whereas FORM is aimed at the academic audience.

Table 2. Evaluation results.

	FORM	**SPLIT**	**CoPAM**
Context			
Specific goal	Capture commonalities and differences of applications in a domain, develop domain architectures and components.	Integrate requirements, architecture and software components for product lines using a defined PL process.	Develop software-intensive product families through component-based architecture.
Method inputs	User's requirements, domain knowledge	Requirement specifications	Product requirements, organization structure
Method outputs	Domain models and application models, application software	PL requirements, PL architecture, PL software components and variation points on all levels.	Architecture-centric, component-based product families.
User			
Motivation	Not applicable	Not applicable	Not applicable
Needed skills	FORM concepts and notations, ASADAL	UML or OCL, OOAD Aspect analysis technique	ADL, IDL, UML, OOAD
Contents			
Models	Context model, Feature model, Operational models, Architecture model, Component model	Requirements and scope model, Variation model, Decision model (three levels), Architecture model, Process model	Architecture models, Component models, Business model, Process model
Method structure	1. Domain engineering: Feature modeling, architecture modeling, Component engineering 2. Application engineering	1. Domain engineering: Analysis (requirements) Design (architecture and components) Implementation (code)	1. Domain engineering: Product family engineering, Platform Engineering 2. Product engineering
Tool support	ASADAL	This issue is still open.	Implicitly defined as a set of tools.
Validation			
Maturity	Based on FODA [13] that has had several applications and extensions after year 1990.	Experimental method, first version, refinement is underway.	Inherits the maturity of several family engineering methods.
Domains	Electronic bulletin board [14], Private Branch Exchange [15], Elevator Control Systems [18]	Ground vehicle simulators Air supervision systems Switch maintenance systems [7].	Telecommunications, consumer electronics and medical imaging systems [2].

In the element of method *contents*, it can be noticed that although software architecture is considered not to include requirements, the FORM and the SPLIT methods

state that an interface between the requirements and architecture design is needed. This interface is a feature model in the FORM and a requirements model in the SPLIT. In spite of the inconsistency in the step names of the three methods, the practical activities done in the steps are often the same. For instance, platform engineering of CoPAM develops a platform of reusable components, which refers to the design of domain components and architecture in FORM and SPLIT. The tool support of methods is often, as now, an open issue. CoPAM defines that, among others, the following kinds of tools are used: word processors, visual modeling tools, code generators, (cross-) compilers, linkers, debuggers and configuration management tools. SPLIT leaves the tool issue open, where the FORM is at the other end by defining a specially developed tool, ASADAL, that supports the FORM explicitly.

In order for a method to prove its suitability for the domain(s) it is designed for, the method has to be validated and mature enough. This aspect is taken into account in the fourth element of evaluation: *validation*. Among the three methods selected to this evaluation, the SPLIT method is the most immature method, whereas the FORM is the most long-lived one, having being applied to several industrial case studies on various domains. CoPAM is surely the first of its kind, but it comprises existing (and therefore mature) family engineering methods, and inherits the strengths of those methods. All the methods have either been validated already or are under validation on various domains. Almost all of these domains can be categorized under the domain of embedded software, except for the electronic bulletin board domain.

Summary

The evaluation was done on the basis of publications concerning the evaluated methods. The Feature Oriented Method for product line software engineering is the oldest of these three methods and is, therefore, rich in publications [12], [14], [15], [18]. The research done with COPA [2] is not yet complete. COPA is derived from CoPAM [3]. SPLIT, for definition and construction of product lines, provides the most limited material for the evaluation [7].

Adopting any of these product line architecture design methods provides several benefits: first, the reuse of proven core assets and consistency of development lead to risk reduction; second, the consistency of development reduces the amount of new development. Therefore, the final results of adopting a software product line are an increased number of products and a decreased amount of rework.

However, the reasons mentioned above do not necessarily motivate the real user of the product line architecture design method, i.e. the architect. High motivation of method users is essential in order to apply a PLA method successfully and reach the full benefits. Examples of user motivation are, e.g., ease of learning, ease of documentation and increased understanding among the development team through unified method, notation and tool(s). From the method users' point of view, most of these methods do not highlight the importance of the tool issue enough. In addition to tool support, the current PLA design methods should clarify the ordering or dependencies between the design steps, models and diagrams.

References

1. ASADAL http://selab.postech.ac.kr/realtime/public_html. [Referenced 7.1.2002].
2. America, P., Obbink, H., Muller, J. & van Ommering, R.: COPA: A Component-Oriented Platform Architecting Method for Families of Software Intensive Electronic Products. Tutorial in SPLC1. August 28-31. Denver, Colorado (2000)
3. America, P., Obbink, H., van Ommering, R., van der Linden, F.: CoPAM: A Component-Oriented Platform Architecting Method Family for Product Family Engineering. In: Donohoe, P. (ed.): Proceeding of the SPLC1. Massachusetts, Kluwer Academic (2000) 167 - 180
4. Bachmann, F., Bass, L., Chastek, G., Donohoe, P., Peruzzi, F.: The Architecture Based Design Method. CMU/SEI, Technical report, 2000-TR-001 (2000)
5. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley (1998)
6. Bass, L., Klein, M., Bachmann, F.: Quality Attribute Primitives and the Attribute Driven Design Method. In: Proceeding of the PFE-4 (2001) 163 - 176
7. Coriat, M., Jourdan, J., Boisbourdin, F.: The SPLIT Method, Building Product Lines for Software-Intensive Systems. In: Donohoe, P. (ed.). Proceeding of the SPLC1. Massachusetts, Kluwer Academic Publishers. (2000) 147 - 166
8. Forsell, M., Halttunen, V., Ahonen, J.: Evaluation of Component-Based Software Development Methodologies. In: Penjam, J. (ed.). Proc. FUSST'99. Tallinn, Estonia. Institute of Cybernetics at TTU (1999)
9. Hofmeister, C., Nord, R., Soni, D.: Applied Software Architecture. Addison-Wesley (2000)
10. Jaaksi, A., Aalto, J-M., Aalto, A., Vättö, K.: Tried & True Object Development: Industry-Proven Approaches with UML. Cambridge University Press (1999)
11. Jayaratna, N.: Understanding and Evaluating Methodologies. McGraw-Hill Book Company (1994)
12. Kang, K. C. A Feature-Oriented Method for Product Line Software Engineering. In: The First Software Product Line Conference, SPLC1. Tutorial 2 (2000)
13. Kang, K. C., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-Oriented Domain Analysis (FODA) feasibility study. CMU/SEI. Technical Report. 90-TR-21 (1990)
14. Kang, K. C., Kim, S., Lee, J., Kim K., Shin E. Huh, M.: FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. Annals of Software Engineering. Vol. 5. (1998) 143 - 168
15. Kang, K. C., Kim, S., Lee, J., Lee, K.: Feature-Oriented Engineering of PBX Software for Adaptability and Reusability. Software Practice and Experience 29(10). John Wiley & Sons Ltd. (1999) 875 - 896
16. Kronlöf, K.: Method Integration: Concepts and Case Studies. Wiley (1993)
17. Kruchten, P. 4+1 View Model of Software Architecture. IEEE Software. Vol. 12, No. 6. (1995) 42 - 50
18. Lee, K., Kang, K. C., Koh, E., Chae, W., Kim, B., Choi, B. W. Domain-Oriented Engineering of Elevator Control Software: A Product Line Practice. Donohoe, P. (ed.). Proceeding of the SPLC1. Massachusetts, USA. Kluwer Academic Publishers. (2000) 3 - 22