Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA and QADA

Mari Matinlassi
VTT Technical Research Centre of Finland, P.O Box1100, 90571-Oulu FIN
Mari.Matinlassi@vtt.fi

Abstract

Product line architectures (PLAs) have been under continuous attention in the software research community during the past few years. Although several methods have been established to create PLAs there are not available studies comparing PLA methods. Five methods are known to answer the needs of software product lines: COPA, FAST, FORM, KobrA and QADA. In this paper, an evaluation framework is introduced for comparing PLA design methods. The framework considers the methods from the points of view of method context, user, structure and validation. Comparison revealed distinguishable ideologies between the methods. Therefore, methods do not overlap even though they all are PLA design methods. All the methods have been validated on various domains. The most common domains are telecommunication infrastructure and information domains. Some of the methods apply software standards; at least OMG's MDA for method structures, UML for language and IEEE Std-1471-2000 for viewpoint definitions.

1. Introduction

Software product lines (PL) are a well-known approach in the field of software engineering. Several methods have been published to address the problems of PL engineering. Methods are diverging in terminology and application domains. Therefore it is difficult to find out the differences and similarities of the methods.

Only few attempts have been made to evaluate or compare the product line architecture (PLA) design methods, e.g. in [1], [2] and [3]. Lopez-Herrejon and Batory propose a standard example case for evaluating product line methods. However, this example is very close to implementation and measures method features with performance benchmarking of the products the method outputs. This kind of evaluation of product line methods is very limited and a comparison covering also the other aspects of PL methods is required. The other example of surveys on product line architectures touches all the aspects related to the product line from assessment to domain engineering and testing. However, this report either does not provide any comparisons that would concern product line design methods. The third attempt represents a covering survey on software architecture *analysis* methods however, software architecture *design* methods are not considered.

On the basis of our studies, there are five methods answering the needs of product lines from the software architectural point of view. In alphabetical order they are COPA[4], FAST[5], FORM[6], KobrA[7] and QADA[8].

The first of the methods mentioned, a Component-Oriented Platform Architecting Method for product family engineering, i.e. COPA, is a component-oriented but architecture-centric method that enables the development of software intensive product families. FAST – Family-Oriented Abstraction, Specification and Translation - is a software development process focused on building families. Feature-Oriented Reuse Method for product line software engineering, FORM is an extension to the FODA [9] method. The core of FORM lies in the analysis of domain features and the use of these features to develop reusable and adaptable domain artifacts. That is, FORM is a feature-oriented approach to product line architecture engineering. Kobra again is an acronym for Komponentenbasierte Anwendungsentwicklung, denoting a practical method for component-based product line engineering with UML. Quality-driven Architecture Design and Analysis, shortly QADA states a product line architecture design method providing traceable product quality and design time quality assessment.

The purpose of this investigation was to study and compare the existing methods for the design of software product line architectures. The intention of this paper is not to provide an exhaustive survey on the area but provide a state-of-the-art of current PLA practices and help others to understand and contrast alternative approaches to product line design. This paper does neither guide in selecting the right approach for PLA design but opens up a basis for creation of such a decision tool. First, this paper provides background knowledge on architectural design methods and introduces a comparison framework for evaluating PLA design methods. Then, the five PLA design methods are briefly presented and compared against the framework. The most remarkable observations of the comparison close the paper.

Proceedings of the 26th International Conference on Software Engineering (ICSE'04)
0270-5257/04 $20.00 © 2004 **IEEE**

2. Architecture Design

Architectural views have been the basis for a number of techniques developed and used during the last few years for describing architectures. It seems that the first of them was "4+1 views to software architecture" [10]. The four main views used are logical, process, physical and development view. The logical view describes an object model. The process view describes the design's concurrency and synchronization aspect. The physical view describes the mapping of the software onto the hardware reflecting the distributed aspect of the system. The development view describes the software's static organization in its development environment. The '+1' denotes the use-case view consisting of scenarios that are used to illustrate the four views.

Jaaksi et al. [11] suggests a slightly modified version of the 4+1 view technique and ends up with 3+1 views necessary to describe the software architecture. The views are the logical, runtime and development view, plus the scenario view. The 3+1 method applies the Unified Modeling Language (UML) as an architectural description language.

Hofmeister et al. [12] define four views (conceptual, module, execution and code view) that are based on observations done in practice on various domains, e.g. image and signal processing systems, a real-time operating system, communication systems, etc.

Despite the fact that the techniques introduced above are capable and exhaustive in their own way; none of them concerns the product line approach to the architectural design.

Architecture Based Design (ABD) method [13] is a quality driven method for designing the software architecture for a long-lived system at the conceptual level of abstraction. In ABD, the conceptual architecture is a representation of the high-level design choices described with three architectural views. Even though the ABD method has been developed further into a new method called the Attribute Driven Design method, ADD [14], it still does not provide more than a coarse grained high-level, i.e. conceptual architecture as an output. Also the support for product line architecture design in the ABD and in the ADD is mentioned but immature.

Only methods (1) specialized for architecture engineering of software product lines and (2) with sufficient materials were selected for comparison. The product line practices concerned e.g. in [15], namely at least Synthesis, Sherlock and Odyssey-DE, are out of the range of this investigation. In addition, QASAR by Bosch [16] describes the process and lists method artifacts leaving the other aspects of the method hidden. SPLIT by Coriat et al. [17] also has insufficient materials and is out of the scope of the evaluation.

3. An 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 [18]. NIMSAD framework uses the entire problem solving process as the basis of evaluation and it can be used to evaluate methods on any category. According to NIMSAD, there are four essential elements for method evaluation.

Firstly, the method is evaluated from the element of the problem situation, i.e. the method context. The second element is the intended problem solver, i.e. the user of the method. The third element is the problem solving process, i.e. the method itself. The last element brings the three elements together through self-evaluation. Because rare methods consider evaluation of the method context, or user or contents, herein, the method evaluation element is turned to method validation element and it considers the validation of the method in question and validation of method outputs.

In addition to the NIMSAD framework, the definition of a method and its ingredients [19] has influenced the third element of the framework, i.e. the method contents. Kronlöf defines method ingredients as follows: 1) an underlying model, 2) a language, 3) defined steps and ordering of these steps and 4) guidance for applying the method. Because tools help in execution of the methods, they are also considered in the element of method contents. The third source for the evaluation framework is an application of the NIMSAD framework for component-based software development methods [20].

The goal of this evaluation was not to rate the methods but to provide an overview of current PLA engineering methods and find out if - and how - the methods differ in any aspects of the PLA design. Therefore a neutral, common and quite extensive NIMSAD framework for method evaluation was utilized to derive the fundamental element categories for the framework. NIMSAD framework has earlier been applied in evaluation of software engineering methods. This application of the framework on software engineering methods provided the basis for detailed element definition for categories. With various questions this study tries to address e.g. maturity, practicality and scope of the methods to find differences. On the other hand the goal was to study if the methods really have what it takes to call them a method. These elements were considered in the category of 'contents' by questioning if the methods satisfy the definition of a method. Framework elements were refined to cover features special for product line methods (e.g. variability support). Herein, the evaluation of the "artifact" element is excluded because of space limitations.

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

Category	Elements	Questions
Context	Specific goal	What is the *specific* goal of the method?
	Product line aspect(s)	What aspects of the product line does the method cover?
	Application domain(s)	What is/are the application domain(s) the method is focused on?
	Method inputs	What is the starting point for the method?
	Method outputs	What are the results of the method?
User	Target group	Who are the stakeholders addressed by the method?
	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?
	Guidance	How does the method guide the user while applying the method?
Contents	Method structure	What are the design steps that are used to accomplish the method's specific goal?
	Artifacts	What are the artifacts created and managed by the method?
	Architectural viewpoints	What are the architectural viewpoints the method applies?
	Language	Does the method define a language or notation to represent the models, diagrams and other artifacts it produces?
	Variability	How does the method support variability expression?
	Tool support	What are the tools supporting the method?
Validation	Method maturity	Has the method been validated in practical industrial case studies?
	Architecture quality	How does the method validate the quality of the output it produces?

4. Overview of PLA design methods

4.1. COPA

A Component-Oriented Platform Architecting Method for Families of Software Intensive Electronic Products (COPA) is being developed at the Philips Research Labs. The COPA method is one of the results of the Gaudi project [21]. The ambition of the Gaudí project is "to make the art and emerging methodology of System architecture more accessible and to transfer this know how and skills to a new generation of system architects".

The specific goal of the COPA method is to achieve the best possible fit between business, architecture, process and organization. This goal results in the middle name of the COPA method: the BAPO product family approach [22]. The specific goal of architecture design is to find a balance between component-based and architecture-centric approaches [23], wherein the component-based approach is a bottom-up approach relying on composition. The architecture-centric approach is a top-down approach relying on variation.

COPA covers the following aspects of product lines: business, architecture, process and organizational aspects. Herein, our evaluation concentrates on the architecture and process aspects. According to [24], the application domains of the COPA method are telecommunication infrastructure systems and the medical domain. In addition, the case studies on the consumer electronics domain are discussed in [4] and [25]. Within these domains, COPA assists in building product populations [23]. Product populations denote the large-scale diversity in a product family developed with a component-driven, bottom-up, partly opportunistic software development using, as much as possible, available software to create products within an organization.

Originally, the COPA method starts by analyzing the customer needs. To be more specific, the inputs of the method's architecting phase are facts, stakeholder expectations, (existing) architecture(s) and the architects(s) intuition. The completely applied COPA method produces the final products. To be more specific, the phase of "architecting" aims to produce a lightweight architecture (see [26] for definition) as an output. A lightweight architecture denotes guidelines for architecture more than traditional software decomposition.

COPA is an extensive method targeted to all interest groups of a software company. Especially, the architecture stakeholders of the COPA method are the customers, suppliers, business managers and engineers [27]. The "multi-view" architecting is addressed for these four main stakeholders [26]. Motivation to use COPA is a promise to manage size and complexity, obtain high quality, manage diversity and obtain lead time reduction.

4.2. FAST

David Weiss introduced a practical, family-oriented software production process in the early 1990's. The process is known as the Family-Oriented Abstraction,

Specification, and Translation process. At the time of writing the book on FAST (1999), the process was in use at Lucent Technologies and the evolution was continuing. The FAST [5] process is an alternative to the traditional software development process. It is applicable wherever an organization creates multiple versions of a product that share significant common attributes, such as common behavior, common interfaces, or common code.

The specific goal of FAST is to make the software engineering process more efficient by reducing multiple tasks, by decreasing production costs, and by shortening the marketing time.

Considering the product line aspects, the FAST method defines a full product line engineering process with activities and artifacts. FAST divides the process of a product line into three sub processes, i.e. domain qualification, domain engineering and application engineering [5].

FAST has been applied successfully at Lucent Technologies at least on the domains of telecommunication infrastructure and real-time systems.

Domain qualification starts from receiving the general needs of business line by distinguishing between two cases: one in which you pay little or no attention to domain engineering, and a second one in which you engineer the domain with the intent of making production of family members more efficient. Application engineering starts when application engineers receive the requirements from customers.

Domain qualification outputs an economic model to estimate the number and value of family members and the cost to produce them. Domain engineering generates a language for specifying family members, an environment for generating family members from their specifications, and a process for producing family members using the environment. Application engineering generates family members in response to customer requirements as an output.

The FAST method was born in the industry and has a high practical background. Therefore, FAST seems to be aimed at software engineers and designers currently working in the industry. The use of the FAST method is motivated with a desire to alleviate the problems that make the software developers' task a lengthy and costly one.

4.3. FORM

Kyo C. Kang and his co-fellows in Pohang University of Science and Technology, Korea, propose a Feature-Oriented Reuse Method (FORM) [6] as an extension to the Feature-Oriented Domain Analysis (FODA) method [9]. FORM extends FODA to the software design and implementation phases and prescribes how the feature model is used to develop domain architectures and components for reuse.

FORM has a specific goal on how to apply domain analysis results (commonality and variability) to the engineering of reusable and adaptable domain components with specific guidelines.

The application domain(s) for the FORM method are the telecommunication domain and the information domain. However, the feature exists in any application domain. If the feature model can be obtained from the application domain, FORM can be fit to the needs of other specific domains.

FORM starts with feature modeling to discover, understand, and capture commonalities and variabilities of a product line. Domain engineering starts from the beginning of the software development: context analysis. The primary input is the information on systems that share a common set of capabilities and data.

Domain engineering creates the feature model, reference architecture, and reusable components as an output. Application engineering creates the application software after features have been selected from the feature model, application architecture has been selected from reference architecture and reusable components have been selected from reusable components.

FORM is targeted to the wide spectrum of domain and application engineering, including the development of reusable architectures and code components. It is used at software engineering in many industrial aspects.

The model that captures commonalities and differences is called a "feature model". The use of features is motivated by the fact that customers and engineers often speak of product characteristics in terms of "features the product has and/or delivers". Features are abstractions that both customers and developers understand and should be the first class objects in software development.

4.4. KobrA

Fraunhofer IESE has been developing the KobrA method [28], [29], [30] that is a methodology for modeling architectures. The method stands for Komponentenbasierte Anwendungsentwicklung that is German for "component-based application development" [7].

KobrA denotes itself as a component-based incremental product line development approach or a methodology for modeling architectures. It is also designed to be suitable for both single system and family based approaches in software development. In addition, the approach can be viewed as a method that supports a Model Driven Architecture (MDA) [31] approach to software development, in which the essence of a system's architecture is described independently of platform idiosyncrasies. Another important goal is to be as concrete and prescriptive as possible and make a clear

distinction between the products (i.e. artifacts) and processes.

KobrA defines a full product line engineering process with activities and artifacts. The most important parts of PL engineering are framework engineering and application engineering with their sub steps, but KobrA also defines implementation, releasing, inspection and testing **aspects** of product line engineering process.

KobrA is developed for the information systems domain (i.e. library system in [32]). However, different application domains demand different methodical support. Therefore, KobrA can be customized to better fit the needs of a specific project. The method provides support for being changed in terms of its processes and products. In addition to the application domain, the factors influencing the KobrA method are organizational context, project structure and the goals of the project.

Framework engineering starts from the very beginning of the software development: context realization. Framework engineering does not need any other input than the idea of a new framework with two or more applications.

The other main activity of the method - application engineering - starts when a customer contacts the software development organization. When such an expression of interest is received, an application engineering project is set up and the context realization instantiation is initiated. This activity equals to the elicitation of user requirements within the scope of the framework.

'Komponent realizations' mean low level designs of software components. However, the process is defined as far as to the implementation and testing phases of the software product.

KobrA is definitely aimed at software engineers and designers currently working in the industry. It is a simple method for developing software and the adoption of the method does not probably express overwhelming challenges for software practitioners today. It also provides an opportunity to get involved in the development of a family of applications and smoothly encourages seeking the benefits of reusing existing assets.

KobrA states it is a simple, systematic, scalable and practical method [7]. Simple here means that a method is as economic as possible with its concepts and the features in a method should be as orthogonal as possible. In addition, a method should separate concerns to the greatest extent possible. Systematic expects that the concepts and guidelines defined in the method should be precise and unambiguous. Also, a method should tell developers what they should do, rather that what they may do. Another feature of the method, that products of a method are strictly separated from the process, also serves in reaching a systematic method. A scalable method provides two aspects of scalability, these being granularity scalability and complexity scalability. The first one means that a method should be able to accommodate large-scale and small-scale problems in the same manner using the same basic set of concepts, whereas fulfillment of the last one refers to incremental application of the method concepts. Practicality requires that a method is compatible with as many commonly used implementation and middleware technologies as possible, particular those that are either de facto or de jure standards.

4.5. QADA

The QADA method is being developed at VTT, the Technical Research Centre of Finland. QADA is an abbreviation for Quality-driven Architecture Design and quality Analysis, a method for both to design and to evaluate software architecture of service-oriented systems.

QADA claims to be a quality-driven architecture design method. It means that quality requirements are the driving force when selecting software structures and, each viewpoint concerns certain quality attributes [33]. Architecture design is combined with quality analysis, which discovers if the designed architecture meets the quality requirements set in the very beginning.

QADA method describes the architectural design part of the software development process, including steps and artifacts produced in each step. It also covers the description language used in the artifacts. It does not cover organizational or business aspects.

Quality-driven design is aimed for middleware and service architecture domains. The case studies cover the design of distributed service platform [8], two kinds of platform services for wireless multimedia applications [34] and the design of wireless multimedia game [35]. In addition, a recent case study on traffic information management system is mentioned in [36]. Quality analysis has been applied to the middleware platform [8], spectrometer controller [37] and terminal software [36].

The method starts with the requirements engineering phase that – even though called requirements engineering - means a link between requirements engineering and software architecture design. The aim is in collecting the "driving ideas of the system and the technical properties on which the system is to be designed" [8]. In addition to functional properties, the quality requirements and constraints of the system are captured as input.

The output of the QADA method is twofold: design and analysis. Design covers software architecture at two abstraction levels: conceptual and concrete. Conceptual architecture covers the conceptual components, relationships and responsibilities, which are intended to be used by certain high level stakeholders related to product line, e.g. product line architects or management. Concrete architecture is closer to the so-called

'traditional' architecture description aimed for software engineers and designers. The QADA method does not produce implementation artifacts.

Analysis provides precious information concerning the quality of the design. Analysis results in feedback of whether the design addresses the quality requirements defined for the system. Analysis may also produce quality feedback about an existing system.

The method users are product line architects and software architects or an architecting team. However, the group of stakeholders that use the method output is much wider. At the conceptual level, the stakeholders include system architects, service developers, product architects and developers, maintainers, component designers, service users, project manager and component acquisition, whereas at concrete level, the architectural descriptions are aimed at component designers, service developers, product developers, testing engineers, integrators, maintainers and assets managers. These groups continue by implementing, testing or maintaining the architecture that is designed.

QADA claims - as do almost all the methods – to be a systematic method and simple to learn. In addition, it is applicable to existing modeling tools [8]. The architecture modeling method also improves communication among various stakeholders [38] and conforms to the IEEE standard for architectural description [39].

5. Comparison Results

5.1. Context

Each of the methods under evaluation is distinguishable concerning the specific goal the method has. All the methods have the same *overall goal*, i.e. produce product line architectures. However, to find a difference, a *specific goal* denotes what point(s) does the method press or highlight in PLA development.

Although e.g. both COPA and KobrA are component-based, the COPA method stands out by combining component-based (i.e. bottom-up) and architecture-centric (top-down) approaches with a novel way. Another top-down approach in addition to COPA is the QADA method. However again, QADA has a diverging goal in combining quality-driven approach with the architecture-centric one. The FAST method expresses itself as a process-driven method, and finally, the FORM method represents well-known feature-orientation to product line engineering.

As a feature-oriented approach, FORM states that it also covers the requirements engineering. The commonality analysis in the FAST method covers the requirements phase extensively. The other methods seem to step aside in this area, except that the QADA method represents an interface between requirements engineering and architecture design however this interface cannot be considered as a systematic approach to gather and analyze product requirements. In addition to requirements engineering, the FORM method covers architecture, implementation and process, as does also the KobrA method. What comes to the other methods, the COPA method is the most complete, covering all the aspects of a product line, whereas FAST captures only the process aspect and QADA extends the method's scope from process aspects to architectural aspects.

The information systems domain is the most popular application domain; three methods altogether, namely KobrA (library system [32]), FORM (electronic bulletin board [40]) and QADA (traffic information management system [36]). In addition to the information system, QADA has been applied in middleware [8], [34], the wireless multimedia domain [35] and in the space application domain [37].

In addition to the electronic bulletin board system, the FORM method has been applied on the elevator control system [41] and the telecommunication infrastructure system [42]. The telecommunication infrastructure domain has been the application domain of also COPA [24] and FAST [5]. The FAST method has been applied on the domain of real-time systems as well [5]. Quite apart from that, the COPA method alone among the methods extends to the medical domain. The COPA case studies on the consumer electronics domain are discussed in [4], [25].

All the methods start from the very beginning, taking context or user requirements as input. While considering the method outputs, all the methods seem to produce quite in-depth outputs by generating results that are close to the implementation. COPA also takes a wider insight into the issue by considering the business and organizational aspects. KobrA defines the process as far as to the implementation and testing phases of the software product. Furthermore, the QADA method is distinguished with output information concerning the quality of the design.

5.2. User

The users of the method are either people who actually use the method, i.e. follow the steps and create the defined artifacts, or people who benefit and use the outputs of the method. It seems the methods agree on the rough division of stakeholder groups related to product line engineering: engineers, architects, business managers and customers. To make a difference, KobrA perhaps is the most practical method aimed at software engineers and designers currently working in the industry. It is a simple method for developing software, and the adoption of the method does not probably express overwhelming challenges for software practitioners today. The conformance to a language standard (UML) and usage of commercial tools emphasizes the practicality and

applicability of the KobrA method. Quite the contrary one may say that FORM is aimed at the academic audience.

What comes to the motivation, adopting any of these product line architecture design methods provides several benefits e.g. reuse, complexity management, higher quality and shorter time-to-market. However, these benefits do not motivate the real method users (software architects) as well as the following implicit reasons. Both KobrA and QADA are developed with a goal to produce a simple and systematic method. They also conform to commonly known standards: UML (KobrA), MDA (KobrA [7] and QADA [38]) and IEEE-Std-1471-2000 (QADA [33]). With an industry proven background COPA is a practical method, and with extensive architectural descriptions, it improves communication among various stakeholders of PL engineering. As well, feature-orientation of FORM gives a common language and therefore improves communication between customers and engineers.

Considering the question of what are the skills the method users need when applying the method, the following issues were concluded. One of the essential method properties is the method language. Two of the methods have a special notation language or ADL to learn (see [6] for FORM notation and COPA Koala [43]) and the most of the methods apply UML as description language. However, current commercial UML tools do not provide a sufficient customization aspect to the needs of architectural descriptions and therefore, every one of the methods need special or extended tool support. This will scale up the effort needed to learn the method. Furthermore, each method has its own method ideology needed to learn. However a 'skill' needed for this purpose is just an open mind.

All the methods provide descriptive case studies. In addition, FORM provides a special guideline [44] for using a feature-oriented approach. COPA and QADA suffer a lack of method documentation, whereas FAST and KobrA are captured in extensive manuals.

5.3. Contents

FORM, FAST and KobrA define a quite similar structure for the method. The basic idea is to first define the context of the system. After that the main two phases are (1) domain engineering and (2) application engineering. Domain engineering is also called product family engineering or framework engineering and it analyses the commonalities and variabilities among requirements and defines the domain architecture or a component framework. Application engineering instantiates the architectural model from domain architecture and produces application realization. In addition to these two main phases, the COPA method introduces the third phase called platform engineering. Platform engineering focuses on the development, maintenance and administration of reusable assets within the platform. Therefore, platform engineering is nothing more than a sub phase derived from domain engineering. Despite, the steps defined in the QADA method are diverging. First, an interface is defined for requirements engineering, which is somewhat compliant to the context analysis. However, design is divided into two phases of conceptual and concrete architecture design. After both design phases, QADA introduces the phase of quality evaluation that assesses the quality of architectural design against defined quality attributes.

FORM and FAST explicitly define support for variability in requirements elicitation, whereas the other methods do not. In addition, through tool support [45] FORM provides automatic transformation from the requirements to an instance of the domain architecture. The other methods concentrate on capturing variability with graphical language in architectural design. QADA and KobrA content themselves with adapted UML and manual transformation to code, whereas COPA has developed its own language and tools to represent variability and transform component descriptions automatically into code skeletons.

FAST does not define explicit tool support. Instead, Process and Artifact State Transition Abstraction (PASTA) process modeling tool of FAST serves to explain FAST in more detail, to help the user to improve FAST and to help the user to develop automated support for FAST. Quite contrary, the FORM method has a single tool, ASADAL [45] supporting all the features mentioned in [40]. Concerning the rest of the methods, they all mention a set of tools (Table 2).

Table 2. Comparing sets of tools in COPA, KobrA and QADA.

Tool	Method		
	COPA	KobrA	QADA
Koala compiler (special code generator)	X		
KoalaMaker (produces a makefile)	X		
Commercial code editor	X		
Commercial UML tool		X	X
Plug-ins for code editor	X		
Visio (with special stencil)	X		X
Word processing tool		X	X
Configuration management		X	

Only two of the methods (COPA and QADA) apply views/viewpoints. Also FORM mentions three architectural models (also called viewpoints): subsystem, process and module [6]. However, an architectural view/viewpoint [39] is a far broader concept than just a model and closely related to various stakeholders of PLA.

The COPA method refines architecture into five views: customer, application, functional, conceptual and realization views [24], [46]. When looking at the view descriptions (Table 3) it is seen that COPA views are more oriented on describing the whole product than just software architecture. The first two views are so called "commercial" views and the last two are technical views. The Functional view in the middle is both commercial *and* technical [27]. However, "no attempt has been made yet to map out the collected viewpoints on the IEEE [39] ontology [27]". Instead, QADA viewpoints (Table 4) conform to the IEEE standard viewpoint description and provide various viewpoints on the software architecture of the system. The four viewpoints are provided at two levels of abstraction. The difference between the levels is partly also in the aggregation dimension (see [47] for definitions of architectural dimensions). Abstraction level means both abstractions with respect to the main architectural concepts of the system and abstraction from the physical world. Therefore a component at the conceptual level is not a software component but more like a logical concept.

Table 3. Introducing COPA views.

View	Description
Customer	Describes the customer's world. Business modeling from the customer's viewpoint.
Application	Describes the applications that are important to the customer. Application modeling. Customer 'how?'
Functional	Captures the system requirements of a customer application. Product 'what?'
Conceptual	Includes the architectural concepts of the system. Product 'how?' Component identification and aspect design
Realization	Describes the realization technologies for building the system. Product 'how?' "Implementation is part of the realization view [48]"

Table 4. Introducing QADA viewpoints.

Viewpoint	Description
Structural	Structures involved in particular functional or/and quality responsibilities. Quality analysis [36].
Behavior	Dynamic actions of, and within a system, their ordering and synchronization. Analysis of execution qualities [36].
Deployment	Structures are deployed into processes and/or physical computing units. Analysis of execution qualities.
Development	Organizing the design work, describes the technological choices made upon standards, software realization asset management

5.4. Validation

All of the methods have been validated in practical industrial case studies. The COPA method was born in the industry and therefore, perhaps, has the strongest industrial experience with software applications in large product families.

Most of the methods i.e. FORM, FAST, COPA and KobrA ensure quality attributes with non-architectural evaluation methods, such as model checking, inspections and testing. Although KobrA also proposes scenario-based architecture evaluation (SAAM [49]) for ensuring maintainability, none of these methods define an explicate way to validate the output from the domain of application engineering. Despite this, the QADA method has an exceptional way of evaluating software architecture designs before implementation. The quality of the design is validated with a scenario based evaluation method in two phases: conceptual and concrete [8].

6. Conclusions

This study has compared five methods for product line architectural design: COPA, FAST, FORM, KobrA and QADA according to specially developed question framework. The comparison largely rested on the available literature. Based on the combined experience of the five product line engineering methods, the most important conclusions were as follows.

The methods do not seem to compete with each other, because each of them has a special goal or ideology. All the methods highlight and follow this ideology throughout the method descriptions.

- COPA. Concentrated on balancing between top-down and bottom-up approaches and covering all the aspects of product line engineering i.e. architecture, process, business and organization.
- FAST. Family oriented process description with activities, artifacts and roles. Therefore, it is very adapting but not applicable as it is.
- FORM. Feature-oriented method for capturing commonality inside a domain. Extended also to cover architectural design and development of code assets.
- KobrA. Practical, simple method for traditional component-based software engineering with UML. Adapts to both single systems and family development.
- QADA. Concentrated on architectural design according to quality requirements. Provides support for parallel quality assessment of product line software architectures.

The most popular domains for applying the methods have been information and telecommunication (infrastructure) domains. These domains have six case studies published all together. However, also the real-time domain, wireless services, middleware, medical systems and consumer electronics domains have been on trial.

All the methods agree that none of the available commercial tools alone and/or without extensions support product line architectural design. Therefore, special tools or tool extensions have been developed to form a set of tools. This way, product line methods may have a full, practical tool support.

There are not available de jure standards for product line architecture development. KobrA and QADA apply other software standards - namely OMG MDA and UML and IEEE Std-1471-2000 – which provide support for formalizing PLA design.

The aim of this study was to provide a comparative analysis and overview on the PLA engineering methods. In addition, this study may provide a basis for developing a decision tool for selecting an appropriate PLA engineering practice. Meanwhile, getting familiar with all the approaches before embarking on suitable PLA development method is recommended.

Acknowledgements
I want to thank Philips, Avaya Labs, Pohang University, IESE and VTT on their constructive criticism. I also appreciate the help of Mr. Hailang Zuo in data collecting and want to thank Professor Eila Niemelä for providing valuable comments during the work.

7. References

[1] R. Lopez-Herrejon and D. Batory, "A Standard Problem for Evaluating Product-Line Methodologies," in the *Proc. of Generative and Component-based Software Engineering: Third International Conference, GCSE 2001*, vol. 2186, *Lecture Notes in Computer Science*. Springer Verlag, Berlin Heidelberg, 2001.

[2] M. Harsu, "A Survey of Product-Line Architectures," Tampere University of Technology, Report 23, March 2001.

[3] L. Dobrica and E. Niemelä, "A Survey on Software Architecture Analysis Methods," *IEEE Transactions on Software Engineering*, vol. 28, 2002, pp. 638-653.

[4] P. America, H. Obbink, J. Muller, and R. van Ommering, "COPA: A Component-Oriented Platform Architecting Method for Families of Software Intensive Electronic Products,". Denver, Colorado: The First Conference on Software Product Line Engineering, 2000.

[5] D. Weiss, C. Lai, and R. Tau, *Software product-line engineering: a family-based software development process*. Addison-Wesley, Reading, MA, 1999.

[6] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures," *Annals of Software Engineering*, vol. 5, 1998, pp. 143 - 168.

[7] C. Atkinson et al., *Component-based product line engineering with UML*. Addison-Wesley, London, New York, 2002.

[8] M. Matinlassi, E. Niemelä, and L. Dobrica, "Quality-driven architecture design and quality analysis method, A revolutionary initiation approach to a product line architecture," VTT Technical Research Centre of Finland, Espoo, 2002.

[9] K. C. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-Oriented Domain Analysis. Feasibility study,," Software Engineering Institute, Pittsburgh CMU/SEI-90-TR-21, 1990.

[10] P. Kruchten, "The 4+1 View Model of Architecture," *IEEE Software*, vol. 12, 1995, pp. 42-50.

[11] A. Jaaksi, J.-M. Aalto, A. Aalto, and K. Vättö, *Tried & True Object Development: Industry-Proven Approaches with UML*. Cambridge University Press, Cambridge Univ., 1999.

[12] C. Hofmeister, R. Nord, and D. Soni, *Applied Software Architecture*. Addison-Wesley, Reading, MA, 2000.

[13] F. Bachmann, L. Bass, G. Chastek, P. Donohoe, and F. Peruzzi, "The Architecture Based Design Method," CMU/SEI, Technical report 2000-TR-001, 2000.

[14] L. Bass, M. Klein, and F. Bachmann, "Quality Attribute Primitives and the Attribute Driven Design Method," in *4th International Workshop on Software Product-Family Engineering*, F. van der Linden, Ed. Springer, Berlin Heidelberg, 2002, pp. 163 - 176.

[15] C. Kuloor and A. Eberlein, "Requirements Engineering for Software Product Lines," in the *Proc. of the 15th International Conference of Software and Systems Engineering and their Applications (ICSSEA'2002)*, vol. 1. Conservatoire National de Arts et Métiers, Paris, 2002.

[16] J. Bosch, *Design and use of software architectures: adopting and evolving a product-line approach*. Addison-Wesley, Harlow, 2000.

[17] M. Coriat, J. Jourdan, and F. Boisbourdin, "The SPLIT Method, Building Product Lines for Software-Intensive Systems," in the *Proc. of the First Software Product Lines Conference*, P. Donohoe, Ed. Kluwer Academic Publishers, Boston, 2000, pp. 147 - 166.

[18] N. Jayaratna, *Understanding and evaluating methodologies: NIMSAD: a systematic framework*. McGraw-Hill, London, 1994.

[19] K. Kronlöf, *Method Integration: Concepts and Case Studies*. John Wiley & Sons, Chichester, 1993.

[20] M. Forsell, V. Halttunen, and J. Ahonen, "Evaluation of Component-Based Software Development Methodologies," in *Proceedings of FUSST'99*, J. Penjan, Ed. Institute of Cybernetics at TTU, Tallinn, 1999.

[21] Philips, http://www.extra.research.philips.com/natlab/sysarch/GaudiProject.html

[22] R. van Ommering, "Building Product Populations with

Software Components," in *Proceedings of ICSE'02*. ACM, 2002, pp. 255 - 265.

[23] R. van Ommering and J. Bosch, "Widening the Scope of Software Product Lines - From Variation to Composition," in *The Proc. of the Second Product Line Conference, SPLC2*, vol. 2379, *Lecture Notes in Computer Science*, G. Chastek, Ed. Springer-Verlag, Berlin, Heidelberg, 2002, pp. 328 - 347.

[24] J. Wijnstra, "Critical Factors for a Successful Platform-Based Product Family Approach," in the *Proc. of the Second Product Line Conference SPLC2*, vol. 2379, *Lecture Notes in Computer Science*, G. Chastek, Ed. Springer-Verlag, Berlin Heidelberg, 2002, pp. 68 - 89.

[25] R. van Ommering, "Building Product Populations with Software Components," in the *Proc. of the ICSE'02*. ACM, 2002, pp. 255 - 265.

[26] G. Muller, "Light Weight Architecture: the way of the future?," Embedded Systems Institute, Article written as part of the Gaudí project 18th March 2003.

[27] G. Muller, "A Collection of Viewpoints," Philips Research, 2001.

[28] C. Atkinson, J. Bayer, and D. Muthig, "Component-Based Product Line Development. The KobrA Approach," in the *Proc. of the First Software Product Lines Conference (SPLC1).* P. Donohoe, Ed. Kluwer Academic Publishers, Boston, 2000, pp. 289 - 309.

[29] C. Atkinson, J. Bayer, O. Laitenberger, and J. Zettel, "Component-based Software Engineering: The KobrA Approach," 22nd International Conference on Software Engineering (ICSE2000), International Workshop on Component-Based Software Engineering, Limerick, Ireland, June 5-6, 2000 2001.

[30] C. Atkinson and D. Muthig, "Component-based product-line engineering with the UML (tutorial)," in the Proc. of the 7th International Conference on Software Reuse, Berlin2002.

[31] D. Frankel, *Model Driven Architecture, Applying MDA to Enterprise Computing*. Wiley Publishing Inc., Indianapolis, Indiana, 2003.

[32] J. Bayer, D. Muthig, and B. Göpfert, "The Library System Product Line - A KobrA Case Study," Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Technical Report IESE-Report No. 024.01/E, 2001.

[33] A. Purhonen, E. Niemelä, and M. Matinlassi, "Viewpoints of DSP Software and Service Architectures," *Journal of Systems and Software*, vol. 69, 2004, pp. 57 - 73.

[34] A. Tikkala and M. Matinlassi, "Platform services for wireless multimedia applications: case studies," in the *1st International Conference on Mobile and Ubiquitous Multimedia*, Oulu, Finland, 2002, pp. 76 - 81.

[35] P. Lago and M. Matinlassi, "The WISE Approach to Architect Wireless Services," in *Proceedings of the 4th International Conference in Product Focused Software Process Improvement, PROFES2002*, *Lecture Notes in Computer Science*, M. Oivo and s. Komi-Sirviö, Eds. Springer, Berlin, Heidelberg, 2002, pp. 367 - 382.

[36] M. Matinlassi and E. Niemelä, "The Impact of Maintainability on Component-based Software Systems," in the *Proc. of the 29th Euromicro Conference*. IEEE Computer Society, Antalya, Turkey, 2003, pp. 25 - 32.

[37] Dobrica Liliana and N. Eila, "Attribute-based product-line architecture development for embedded systems," in the *Proc. of the 3rd Australasian Workshop on Software and Systems Architectures*. IEEE, Sydney, 2000, pp. 76 - 88.

[38] M. Matinlassi and J. Kalaoja, "Requirements for Service Architecture Modeling," in *Workshop of Software Modeling Engineering of UML2002*. Dresden, Germany, 2002.

[39] IEEE, "IEEE Recommended Practice for Architectural Descriptions of Software-Intensive Systems," *Std-1471-2000*. New York: Institute of Electrical and Electronics Engineers Inc., 2000.

[40] K. C. Kang, "A Feature-Oriented Method for Product Line Software Engineering," Denver, Colorado: The First Software Product Lines Conference, 2000.

[41] K. Lee, K. C. Kang, E. Koh, W. Chae, B. Kim, and B. W. Choi, "Domain-Oriented Engineering of Elevator Control Software: A Product Line Practice," in *Software Product Lines, Experience and Research Directions*, P. Donohoe, Ed. Kluwer Academic Publishers, Boston, 2000, pp. 3 - 22.

[42] K. C. Kang, S. Kim, J. Lee, and K. Lee, "Feature-Oriented Engineering of PBX Software for Adaptability and Reusability," *Software Practice and Experience*, vol. 29, 1999, pp. 875 - 896.

[43] R. van Ommering, F. van der Linden, J. Kramer, and J. Magee, "The Koala component model for consumer electronics software," *IEEE Computer*, vol. 33, 2000, pp. 78 - 85.

[44] K. Lee, K. C. Kang, and J. Lee, "Concepts and Guidelines of Feature Modeling for Product Line Software Engineering," in the *Proc. of the 7th International Conference on Software Reuse*, *LNCS 2319*, C. Gacek, Ed. Springer-Verlag, Berlin Heidelberg, 2002, pp. 62 - 77.

[45] ASADAL, http://selab.postech.ac.kr/realtime/public_html/index.html

[46] P. America, H. Obbink, and E. Rommes, "Multi-View Variation Modeling for Scenario Analysis," in *PFE-5: Fifth International Workshop on Product Family Engineering*, F. van der Linden, Ed. Springer, 2003.

[47] L. Bratthall and P. Runeson, "A Taxonomy of Orthogonal Properties of Software Architecture," in the *Proc. of the Second Nordic Software Architecture Workshop*, 1999.

[48] G. Muller, "Software Reuse; Caught between strategic importance and practical feasibility," Embedded Systems Institute, Article as part of the Gaudí project, 19th March 2003.

[49] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Addison-Wesley, Reading, Massachusetts, 1998.