Journal of Software Engineering and Applications, **2012, 5, 472-476**
http://dx.doi.org/10.4236/jsea.2012.57054 Published Online July 2012 (http://www.SciRP.org/journal/jsea)

Quality-Oriented Software Product Line Architecture Design

Lei Tan, Yuqing Lin, Huilin Ye

School of Electrical Engineering and Computer Science, University of Newcastle, Callaghan, Australia.
Email: lei.tan@uon.edu.au, {yuqing.lin, huilin.ye}@newcastle.edu.au

Received April 10th, 2012; revised May 17th, 2012; accepted May 28th, 2012

ABSTRACT

Software architecture design is a critical step of software development. Currently, there are various design methods available and each is focusing on certain perspective of architecture design. Especially, quality-based methods have received a lot of attentions and have been well developed for single system architecture design. However, the use of quality-based design methods is limited in software product line (SPL) because of the complexity and variabilities existing in SPL architecture. In this paper, we introduce an extra view to the Quality-Driven Architecture Design and Quality Analysis (QADA) method, in order to provide a more effective quality-based architecture design framework for SPL. In this framework, the quality attributes of a software system will be taken into account in the early stage of architecture design and the reference architecture of SPL will be elicited based on quality-related consideration.

Keywords: Software Architecture; Software Product Line (SPL); Quality-Oriented; Requirement Traceability

1. Introduction

The software architecture is the structure of a software system. It contains software elements, visible properties of these elements, and the relationships among them [1]. The visible properties refer to the behaviors of elements such as providing services, performance characteristics, fault handling, and so on. The architecture of a system is the overall abstraction to illustrate how software elements relate to each other and how do they work together. Software architecture is able to scope overall system, to provide a blueprint for system construction.

Software architecture design is one of the most important phases in the software development life cycle, so it attracts a lot of attentions. It is a comprehensive process involving multiple activities. For instance, system decomposition, specifying structural components, and quality attributes tradeoff. A good software architecture is able to address products requirements sufficiently and leads to less costly implementations.

The challenges of architecture design are: 1) how to evaluate the quality of software architecture and how to design high quality architectures; 2) how to cope with non-functional requirements and quality restrictions. There are many methods have been proposed to deal with the first challenge, but developing methods for quality requirements realization are more difficult. Without considering software architecture qualities at design stage could bring negative impact to the later product development due to economic concerns, such as time to market and return on investment.

The architecture design methods are to explore the design options from different perspectives. For example, the "4 + 1" view model [2] aims to describe software architecture in multiple views. A design concern is a feature or behavior of a system, for instance, performance and security are design concerns of most software systems. The purpose of using multiple views is to separate design concerns and each of the multiple views describes the software system from a specific aspect. Separating concerns is a common approach in computer science to break a complex problem into easier subproblems [3]. The "4 + 1" model contains multiple views, namely, logical view, the process view, the development view and the physical view. In the logical view, system is decomposed into a set of key abstractions. The principles of system abstraction, encapsulation, and inheritance are exploited in this view. The process view deals with the issues of system integrity, such as concurrency and distribution. It also specifies how the abstractions from logical view map onto the process view. In the development view, the system is decomposed into subsystems that can be developed by small number of developers. These subsystems are layered as a hierarchy and each layer provides an interface to the upper layer. The physical view is to map the software onto the hardware. The elements identified from previous views need to be mapped onto vari-

ous processing nodes. In addition to the "4" views, scenarios, also known as use cases, are used to put those four views together. These scenarios are abstractions of the most important system requirements. This view-based design concept has been adapted by many other design methods, including COPA [4] and QADA [5]. We also adopt this design concept in our work presented in this paper. There are also some other architecture design methods. For example, ASAAM [6] is an aspectual architecture analysis method to localize design concern, which crosscutting several components, in one architectural component. ADD [7] is an attribute driven method to design software architecture. For more on software architecture design, see details in surveys [8,9].

Non-functional requirements and quality attributes are the properties of software products. They are requirements that have to be satisfied. Quality requirements are used as a bridge to connect business goals and software architectures [10]. In this paper, we focus on quality-oriented SPL architecture design for two reasons: 1) it is hard to evaluate if the architecture design is qualified based on quality criteria; 2) comparing to the functional requirements, qualities attributes are less considered in the design phase. Quality attributes have impact on product line architecture (PLA), product components and the relationships among the components. For example, a design decision based on security concern will affect both encryption and network designs, and the way to balance these two components.

Quality attributes are generally divided into three categories: system qualities, business qualities and architecture qualities. It is crucial to evaluate if qualities requirements have been achieved by architecture design. Realizing the system qualities before developing the concrete system is helpful to stakeholders. The appropriate design methods will model the desired system needs and constraints properly in the early stage. As a result, it is more likely to form the right software architecture.

There are some quality based methods [11,12] for software architecture design. Most of them are scenario-driven. Different from other architectural design methods, they are focus on quality related aspects such as system performance, security of the system and other system qualities. These design methods make sure that the resulting architecture fulfills the quality requirements from stakeholder. However, comparing to other design methods, quality driven design is still in the early development stage overall, especially in SPL engineering.

Software Product Line (SPL) engineering is to develop a collection of related systems which share a common software architecture and related components [13]. The idea of SPL engineering was proposed by Kang *et al.* [14] and the approach has been used in industry over the last decade. The key idea of SPL is to discover commonalities and variabilities across a product family. SPL contains a set of reusable software assets, such as system requirements, source code, and reusable components. These assets are configured and composed to create member products in a product line. A successful SPL is able to improve the development productivity and the quality of the software, significantly reduce cost and time to market.

The architecture of an SPL is to describe both shared components of products family and individual characteristic of single product. The efforts of PLA design are focusing on refining the reference architecture and reusable components. The reference architecture is instantiated and used as a common asset to create concrete architectures for member products. The reference architectures also provide a clear idea of how configurable components can be assembled to form member product architectures.

The remainder of the paper is organized as follows. Section 2 presents and discusses several popular architecture design methods. In Section 3, we describe an extension to QADA method. Section 4 concludes the paper and discusses future works.

2. Software Production Line

Firstly, we briefly summarize some well established SPL architecture design methods, they are FORM [14], FAST [15], COPA [4], KobrA [16] and QADA [5]. These methods have been validated on various domains in industry.

1) *Feature Oriented Method*: Feature-Oriented Reuse Method (FORM) is a feature-oriented approach. FORM is using feature model to realize both commonalities and variabilities of SPL. The result of feature model analysis will be further used to explore the reusable components for a product line.

Two phases are included for reusable architecture development. In domain modeling, features are localized in four layers and each layer presents a level of software development hierarchy. Any instantiation of a feature model is a combination of features from these four layers. The subsystem model defines the overall system structure by packaging service features into subsystems in a distributed environment. Then each subsystem is decomposed into a set of processes considering the operating environment features. Modules are used to create reusable components with specifications to define how to integrate them into applications. In architecture modeling, the reference architecture is defined from three levels of abstractions (subsystem model, process model and module model) by the right feature selections and then pick the proper reusable software components based on concerns from stakeholders.

2) *Process Driven Method*: Family-Oriented Abstraction, Specification and Translation (FAST) is a software development method focusing on building product families. It is intended to enhance the efficiency of development process by reducing multiple tasks, time to market and development cost [17]. This method is suitable in case that one product has multiple versions that have shared elements in common.

Two main activities of FAST are Domain Engineering and Application Engineering. Domain engineering consists of activities to define domain-based family, a modeling language to describe family members. Domain engineering also specifies tools needed for application model analyze. The application engineering is to generate application systems to customer requirements. Each single system, which is defined by the application model, is a member of the family and is generated by tools support developed from domain engineering. In addition to domain engineering and application engineering, FAST has an extra engineering process: domain qualification. Domain qualification is the process to deal with economic related concerns for a domain. During this process, economic related information needs to be specified and included in an economic model as the result.

3) *Component Based Method*: Component-Oriented Platform Architecting (COPA) method is a component-oriented method that enables the development of software intensive product families. COPA method aims for achieving a balance between multiple aspects and concerns. Because it was firstly validated in practical Industry, COPA has the best industrial application experience in large product families.

COPA starts with business phase, which is analyzing the customer needs and expectations. Architecture phase is divided into five views by architectural framework: customer business view, application view, functional view, conceptual view and realization view. COPA also contains three engineering processes within process framework: product family engineering, platform engineering, and product engineering. Product family engineering focuses on the family architecture, platform engineering is dealing with those reusable assets and the responsibility of product engineering is to generate the software products. The architectural activities of product family engineering are to define views on both comercial and technical aspects. The activities of platform engineering are to support product engineering with integration testing and to maintain existing components and platforms. The activities of product engineering consist of developing the specific product components and constructing products.

KobrA method is another component-based product line engineering approach. It is able to be applied on both single system and family based systems. It is a simple and practical method for software development by applying standard UML models and some commercial tools to enhance its applicability.

Each component, is called *Komponent* in KobrA, is described at two levels of abstractions: specification level, which defines *Komponents* behaviors and services, and realization level, which describes how to fulfill the upper level services via lower level *Komponents*. KobrA has two main engineering activities: framework engineering activities and application engineering activities. Framework engineering activity is to create a general framework which includes all product variants of product family. In framework engineering, the specification of a *Komponent* is described by a set of models. Variabilities are determined by decision model whether variabilities can be captured by the existing decisions or it is necessary to add new decisions to the decision model. With the support of a set of models such as interaction model, structural model, activity model, and decision model, it is able to specify how to design a single *Komponent*. The purpose of application engineering activity is to implement the framework in order to derive member products from the family.

4) *Quality Driven Method*: Quality-driven Architecture Design and quality Analysis (QADA) is a traceable quality based method to design and evaluate software architecture. QADA contains scenario-based quality analysis to evaluate if the architecture design options meet the quality requirements.

QADA consists of three viewpoints: structural view, behavior view, and deployment view at two levels of abstractions: conceptual level and concrete level. Quality attributes are categorized to related views and each view has associated targets on the two levels at certain design phase. As a quality-driven method, quality analysis is performed at both levels with different attentions. Analysis on conceptual level focuses on variability analysis and architectural analysis by quality based methodology. Quality analysis of concrete architecture emphasizes on the customer value analysis and scenario-based quality analysis. The purpose of architectural analysis at conceptual level is to provide a knowledge base for a more comprehensive quality attributes analysis at the concrete level.

QADA is the only quality-driven architecture design method among these methods. It contains several views at different levels to separate concerns and it provides a quality-driven link between software requirement and architecture. Quality attributes are very important to architecture development since quality attributes guide the decision on architectural style selections and thus affect the construction of system architecture. In [5], author mentioned that additional views are needed to present more information of quality attributes analysis in future

research. So in this paper, we will extend QADA method by adding an extra view to improve this quality based PLA design method.

3. Extension to QADA

We propose to include a quality view into the multiple views of QADA to improve the traceability of the reference architecture. The purpose of having quality view is to provide a picture of system components and packages and to illustrate how they fulfill the quality requirements of a product line. At the same time, it describes the impact of quality requirements on the reference architecture at the conceptual level.

Quality based frameworks like [1,18] could be modified and used to develop this view. To develop a reference architecture, we start from requirement engineering. Because some quality requirements have more influences on the architectural design than others, so to help to manage the complex requirements specifications and stay focus on the most critical design decisions, it is important to focus on those important quality requirements which have strong impacts on the reference architecture. We call these important quality requirements the architecture development drivers. The architecture drivers need to be embedded into reference architecture and refined for member product architecture development.

A quality scenario is a quality requirement of the associated quality attribute. Quality scenarios are used for identifying architecture drivers. Quality scenarios can be categorized into general scenarios and concrete scenarios. General scenarios are system independent and they can be applied for all member products in a product family. Concrete scenarios are derived from the general scenarios and they are solid and to be used as quality requirements of a specific member system. To develop the reference architecture, we mainly focus on general scenarios. General scenarios can be represented by a table including all possible alternatives of certain quality attributes. The specification of quality scenarios and examples of modifiability-related scenarios are given in **Table 1**.

The next activity of developing quality view is to identify mechanisms [18] for the realization of family architectural drivers, which contain a set of general scenarios. Mechanisms contribute to the quality requirements achievements. A strategy-oriented mechanism, which is related to collection of architecture drivers, describes design strategies to satisfy the architecture drivers. Such mechanisms are abstract, but can be used for both the reference architecture development and member product reference architecture design. A mechanism could also be very specific, contains a set of components, connection types and their responsibilities for fulfilling certain quality requirements. To map quality scenarios onto architecture, system components and the connections, which derived from mechanisms, are used to form the reference architecture and to implement the quality requirements of SPL. Components, contributing to the related architecture drivers, can be grouped, and these are the subsystem options and design options. When developing architecture for specific member product, concrete scenarios will be take into account and all these subsystems will be configured and refined.

Table 1. Quality attributes scenario and an example.

Scenario parts	Specification	Example
Source of stimulus	Where stimulus are from	Users, developers, admins
Stimulus	Conditions when stimulus happen	Add/delete/modify functionalities
Environment	Certain conditions when stimulus occur	Design time, run time
Artifact	Whole or part of system is stimulated	Interface, platform, system
Respond	Activities when stimulus happen	Locate modified place, deploy modification
Respond measure	Measurable to achieve the requirements	Cost, efforts

Table 2 used in [18] is to present how scenarios map onto a product line of vehicle navigation systems. The scenario is that: “recalculate route due to incoming radio data message”. This is a performance-related scenario and it is mapped onto the logical view of architecture. Six components (from two subsystems) are involved in this scenario. The components and their responsibilities are represented in the table.

Functional requirements will be further used to evaluate the capability of the reference architecture. Design options from scenarios mapping contain system components to achieve the functionalities of systems, and then functionalities will match the system requirements by using the appropriate mechanisms.

4. Conclusion and Future Works

We have reviewed software product line architectures design process. Quality attributes play an important role in the process as they have big impacts on software architecture design. The main benefit of quality based design methods is to put stakeholders’ expectations at first place to ensure the quality of software products.

We have introduced the quality view to QADA to improve the traceability of quality based PLA design. The improved framework emphasis on considering how the reference architecture fulfills the quality requirements. In quality view, quality scenarios is mapped onto the reference architecture and corresponding components. The results of applying this framework are the valid reference

Table 2. Scenario mapping example (From [18]).

Design Component	Responsibilities
Radio Data Provider	Receive radio data message
TRS Management	Decode and convert data into internal format accumulate message and replace expired data check data filtering
Atlas	Update street elements
Vehicle Tracking	Report position
Route Management	Check if recalculation is necessary notify interested parties
Route Calculation	Route calculation

architecture options at the conceptual level to satisfy various quality requirements of the SPL.

As the reference architecture will be refined and configured to produce member product architecture, therefore, it is unavoidable that we have to manage the quality attributes tradeoffs. This will be part of our future work. Moreover, this paper only takes system quality attributes into account when designing a reference architecture, other architecture attributes also need to be considered, this will be another task in the future works.

REFERENCES

[1] L. Bass, P. Clements and R. Kazman, "Software Architecture in Practice," 2nd Edition, Addison-Wesley, New York, 2003.

[2] P. Kruchten, "The 4 + 1 View Model of Architecture," *IEEE Software*, Vol. 12, 1995, pp. 42-50. doi:10.1109/52.469759

[3] H. Mcheick and H. Mili, "Understanding Separation of Concerns," *Workshop on Early Aspects—Aspect Oriented Software Development* AOSD04, 2004.

[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," *The* 1*st Conference on Software Product Line Engineering*, Denver, 28-31 August 2000.

[5] M. Matinlassi, E. Niemel 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.

[6] B. Tekinerdogan, "ASAAM: Aspectual Software Architecture Analysis Method," 4*th Working IEEE/IFIP Conference on Software Architecture*, Oslo, 12-15 June 2004.

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

[8] D. Falessi, G. Cantone, R. Kazman and P. Kruchten "Decision-Making Techniques for Software Architecture Design: A Comparative Survey," *Journal of ACM Computing Surveys*, Vol. 43, No. 4, 2011, pp. 1-28. doi:10.1145/1978802.1978812

[9] N. May, "A Survey of Software Architecture Viewpoint Models," *Proceedings of the* 6*th Australasian Workshop on Software and System Architectures*, Melbourne, 29 March 2005, pp. 13-24.

[10] R. Kazman, R. L. Nord and M. Klein, "A Life-Cycle View of Architecture Analysis and Design Methods," *Software Architecture*, *Technical Note*, 2003.

[11] L. Bass, M. Klein and G. Moreno, "Applicability of General Scenarios to the Architecture Tradeoff Analysis Method," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 2001.

[12] R. Kazman, J. Asundi and M. Klein, "Quantifying the Costs and Benefits of Architectural Decisions," *Proceedings of the* 23*rd International Conference on Software Engineering*, Toronto, 12-19 May 2001, pp. 297-306.

[13] J. Bosch, "Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach," Addison-Wesley, New York, 2000.

[14] 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. doi:10.1023/A:1018980625587

[15] D. Weiss, C. Lai and R. Tau, "Software Product-Line Engineering: A Family-Based Software Development Process," Addison-Wesley, Boston, 1999.

[16] C. Atkinson, J. Bayer, C. Bunse, E. Kamsties, O. Laitenberger, R. Laqua, D. Muthig, B. Paech, J. Wust and J. Zettel, "Component-Based Product Line Engineering with UML," Addison-Wesley, Boston, 2002.

[17] M. Matinlassi, "Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA and QADA," *Proceedings of the* 26*th International Conference on Software Engineering*, IEEE Computer Society, Washington, 2004, pp. 127-136.

[18] S. Thiel, "On the Definition of a Framework for an Architecting Process Supporting Product Family Development," 4*th International Workshop on Software Product-Family Engineering*, Springer-Verlag, London, 2002, pp. 125-142.

