Modeling Quality Attributes in Software Product Line Architecture

Lei Tan, Yuqing Lin and Huilin Ye
School of Electrical Engineering and Computer Science
University of Newcastle, Callaghan, 2308, NSW, Australia
Email: lei.tan@uon.edu.au, {yuqing.lin, huilin.ye}@newcastle.edu.au

Abstract—The main task of the software product line architecture design is to develop the reference architecture which represent the base structure of the member products. Quality-driven Architecture Design and quality Analysis (QADA) is the architecture design method, which has attracted many attentions since it emphasizes on addressing quality issues of the member products at early stage of the system development. In this paper, we introduce two additional views (the quality view and the tradeoff view) to the QADA framework. These extra views will improve the requirements traceability, i.e. it is easier to understand how the requirements on software quality are mapped onto the system architecture, thus will help in developing the concrete architecture for the member products. We have also included a case study to illustrate our approach in details.

I. INTRODUCTION

A software architecture is the structure of the software system. It describes the software elements, their characteristics and also how the software elements interact with each other [2]. A qualified software architecture provides a blueprint for system construction and composition, it is a key factor to a successful software development. There are many challenges in software architecture design. For example, modeling the non-functional requirements, especially, those requirements on the quality of the software. Non-functional requirements and quality attributes are the very important parameters of software products. Quality requirements of a system are served as a bridge between business goals and software architectures [10]. Many quality oriented architecture methods have been proposed, for example, ATAM [9] and CBAM [8]. These architecture design methods give extra attentions to the non-functional requirements in the architecture design phase and support explicit mapping of non-functional requirements and qualities of software onto the software components.

Software Product Line (SPL) engineering is about developing a collection of systems which share great commonalities [3]. The idea of SPL was initiated by Parnas [14] and has been further developed by Kang et al [7]. The concept of SPL is to discover both commonalities and variabilities among member products of the product family. SPL will contain a set of reusable software assets, such as system requirements, source code, architectures and documentations. When developing a member product of the family, these reusable assets will be refined, modified and configured, resulting a shortened development cycle. Furthermore, since these reusable components have been tested and successfully used in other products, so it is reasonable to expect the product developed by using these reusable components has better quality than the product developed from scratch. A successful SPL supports systematic software reuse and reduces the development effort, meanwhile, improves the quality of the member products.

The SPL architecture design is different from the traditional software architecture design. One of the main challenges is to deal with the variabilities existing among the member products. More specifically, the member products have different functions and different expected qualities. These differences very often result in quite distinct software architectures for the products. So in SPL architecture design, it is common to design a reference architecture with configurable components. The reference architecture [4] is the software architecture that provides the common structures, components and their relationships to the existing systems in a particular domain or a SPL. The reference architecture and components will be further refined when developing member products architectures. Thus the reference architecture, possibly reference architectures, of a software product line in certain degree model the common structure of the member products. And at the same time, it should be flexible to be modified to derive concrete architecture for member product.

When configuring the reference architecture to produce member product architectures, the quality related refinements are particularly important. It is crucial for the reference architecture to support requirement traceability, i.e. providing explicit information on how the quality requirements are mapped onto the software architecture. This would give the software engineers certain degree of confidence to proceed by knowing that the quality requirements had been considered and modeled in the architecture. Furthermore, since member products have different desired qualities, so the reference architecture should provide architectural options for satisfying quality attributes at different levels, and furthermore, the restrictions and relationships among these options. Not only these information could be used for configuring member product architecture, but also they could be used for product configuration.

The rest of the paper is organized as the follows. Section 2 introduces some existing SPL architecture design methods, then gives a brief summary of quality driven architecture design method - QADA. In Section 3, we introduce two quality associated views as extensions to QADA. We include a case study in Section 4 and Section 5 concludes the paper and
discusses the future works.

II. SOFTWARE PRODUCT LINE ARCHITECTURE METHODS

There are several architecture design methods available for Software Product Line, for example, the Feature-Oriented Reuse Method (FORM) [7]. Feature modelling [6] is a well-known method used in SPL to manage the variabilities of family products. The feature model represents the collection of the requirements on the member products and it is used for developing the SPL architecture. In FORM, features are localized in four layers and each layer represents a level of software development hierarchy. The reference architecture is defined from three levels of abstractions. Modules are used to create reusable components with specifications to defining how to integrate them into applications. The reference architecture will be then refined based on the feature selections of the member product. There are other design methods such as the Family-Oriented Abstraction, Specification and Translation (FAST) [17] and Component-Oriented Platform Architecting (COPA) [1]. For more information about these SPL architecture methods, please see the survey [12].

Among all the current popular design methods, Quality-driven Architecture Design and quality Analysis (QADA) [13] is the only quality oriented architecture design method. The QADA is structured by two levels: conceptual and concrete architecture design. The intention of QADA is to develop the complete PLA and a set of components before developing member products. The main software structural concerns are the quality requirements. QADA introduces scenario-based quality analysis to compare and select the design options to ensure satisfying the quality requirements in the architecture. It also includes the quality evaluation activities to assess the quality of architecture. The significance of QADA is that it is able to provide traceable quality-driven links between software requirement and architecture.

QADA deploys multiple views to describe the software architectures at different levels to separate concerns. Architecture views are developed from the abstract descriptions at the conceptual level down to the concrete level during the architecture design and analysis. Systems’ commonalities and variabilities of a product line will be considered in the structural views at the conceptual level. Different solid quality attributes will be considered within related views at the concrete level.

The quality attributes of the product family are critical factors in design process since they affect the architectural style selections and the architecture constructions. It was mentioned in [13] that it is important to investigate “how to take better advantage of architectural concepts to analyze a software product line for quality attributes in a systematic way”. The author further suggested that additional views are needed to present more information in QADA for quality attributes analysis. So in the next section we extend QADA method by adopting two extra views to help quality based PLA design.

III. EXTENSIONS TO QADA

We enhance the QADA method by introducing two additional views: quality view and tradeoff view. The quality view was developed in our work [15] so we only give a brief introduction in this paper. The proposed quality view is to provide requirement traceability in the reference architecture and help member product configurations. The tradeoff view is to deal with quality attributes tradeoff issues in SPL. A tradeoff decision could cause big changes in system architecture, thus has to be managed properly.

A. Quality View

The purpose of quality view is to provide the picture of system structural packages and then to illustrate how they fulfill the quality requirements of product line. More formally, the impact of quality requirements on the reference architecture at the conceptual level is specified in the quality view. The quality view contains many subsystems. For each quality attribute, the related parts of the reference architecture are identified and included in the subsystems. The subsystems also contain the information of how the components interact each other. Since the components are configurable, it also contains the options for satisfying the quality at different levels. The quality view enables the requirement traceability, especially for the non-functional requirements and the quality attributes. The quality view is very important for the member products configuration and derivation.

As show in Fig. 1, to design the reference architecture, we start from requirement engineering. Those critical quality requirements, so called architectural drivers, will be identified as they have more influence on the architecture than others. Once we have all the architecture drivers, then the mechanisms [16] will be used to map the architectural drivers to the reference architecture. Mechanisms contain one solution or possibly multiple solutions to a given problem. A concrete mechanism contains the actual components, connections and their responsibilities. From these mechanisms, we can identify the reusable components, connections and the common properties of the product line, which then form various subsystems and design options for the reference architecture. For more details of our approach, please see [15].
IV. PACS CASE STUDY

Picture Archiving and Communication System (PACS) [5] is a medical imaging system which contains various tasks in medical applications. A typical PACS contains subsystems such as image acquisition, storage, and display, which are integrated by digital networks. PACS is a quality-sensitive system that associated quality attributes have to be considered in system architecture design. Because of the complexity of the whole system in the context of SPL and limited space, we only include partial quality-related variabilities and components of the system to demonstrate our approach.

A. Quality View

The goal of the quality view is to clarify the relationships between those components and quality attributes. Fig. 3 illustrates the reference architecture with subsystems and components. By matching scenarios and quality factors with system components, the relationships between components and quality factors are identified to be included in the quality view.

B. Tradeoff View

To develop the tradeoff view, we start from quality scenarios. Suppose that we are given two scenarios (S1, S2) with different quality considerations.

TABLE I
QUALITY ATTRIBUTES AND RELATED COMPONENTS (CONNECTORS, PORTS)

From Table I, we could see some of the quality attributes, and the system components, connectors etc contributing to the realization of the qualities attributes. For example, the Expert System component has a big impact on system’s usability and the ports to realize this responsibility could be VPN (Virtual Private Network) or SSL (Secure Socket Layer). Expert System is able to improve efficiency of systems by providing pre-scan and prioritizing, meanwhile to improve the accuracy of system by providing reliable diagnosis recommendations. Moreover, Expert System has the impact on systems’ security considerations because of the issues such as data encryption, network security and authentication. This information is all included in the quality view so it is clear to the developer that the architectural subsystem related to the quality attributes of the software product line.

B. Tradeoff View

To develop the tradeoff view, we start from quality scenarios. Suppose that we are given two scenarios (S1, S2) with different quality considerations.

Fig. 2 shows how we develop the tradeoff view. Quality scenarios from the quality view can be used as input of quality tradeoff process. Quality scenarios are prioritized by both users and domain experts to explore the importance of each scenario. The intention of scenario prioritization is to provide a general idea about how to handle the conflicts between quality requirements. If the conflict existing between two scenarios, the higher priority scenario should be better satisfied than the lower priority ones.

Quality related design decisions affect corresponding components, which have further impact on the structure of the system. Moreover, relationships between quality attributes could be quite complex. For example, high reusability normally leads to weaker reliability, but enhances flexibility and testability of systems. For the relationships and constraints between quality attributes, please see the work by Zulzalil et al [18]. Understanding the relationships is helpful to identify the tradeoff options. The constraints between quality attributes can be identified from the corresponding scenarios. Valid scenarios combinations refer to a collection of corresponding architecture drivers and then further link to the appropriate mechanisms, which lead to the design of a valid architectural option. Conflict scenarios combinations indicate the conflict requirements, which indicate the tradeoff issues. These options of tradeoff generate the architecture recommendations. The tradeoff view consists of multiple sub-views, for each set of related quality attributes, the sub-view contains the corresponding architecture recommendations which summarize valid subsystem alternatives that achieve both system functionalities and quality requirements.

The tradeoff view includes the information of quality-related decisions and functional selections. The tradeoff view can be traced back into the quality view and used for member product configuration. In the next section, we use an example to illustrate how our approach works.

IV. PACS CASE STUDY

Picture Archiving and Communication System (PACS) [5] is a medical imaging system which contains various tasks in medical applications. A typical PACS contains subsystems such as image acquisition, storage, and display, which are integrated by digital networks. PACS is a quality-sensitive system that associated quality attributes have to be considered in system architecture design. Because of the complexity of the whole system in the context of SPL and limited space, we only include partial quality-related variabilities and components of the system to demonstrate our approach.

A. Quality View

The goal of the quality view is to clarify the relationships between those components and quality attributes. Fig. 3 illustrates the reference architecture with subsystems and components. By matching scenarios and quality factors with system components, the relationships between components and quality factors are identified to be included in the quality view.
To protect patients privacy, image data has to be encrypted before store in the data base. Only authorized request is accepted to access data from storage with approved authentication.

To improve the efficiency of system performance, internal request such as Expert System, is able to access raw data from image storage before data encryption.

From these scenarios, we need to abstract the architecture driver which is the information impact the architecture design. For example, in our case, we could identify the needs of encryption, authentication, data backup and data remote access. For each architecture driver, mechanisms give out the recommended solutions. For instance, for encryption, options such as public key encryption and private key encryption will be given by the corresponding mechanisms. Each solution recommended by a mechanism has positive impact to some quality attributes and has negative impact to the other quality attributes. For example, image reconstructors are responsible for calculating 3D images by using a sequence of 2D images. There are two options available: real-time reconstructor and portable reconstructor. Real-time reconstructor is faster but support wide range of third party component applications. So here we have to consider the tradeoff between the system performance and flexibility of the system.

In the tradeoff view, we need to present various of tradeoff options and the impact of each option on the rest of the system.

The conflicts between quality attributes could be quite complicate. We use the tradeoff between “Performance” and “Security” to illustrate how the tradeoffs impact on the reference architecture. Fig. 4 represents two architectural options which have different “Performance” and “Security” levels. In Option1 implements higher security level, all image data have to be encrypted immediately before storage. So the encryption of “Security” component is involved in the first place. “Image Process” component will require the encrypted data through “Security” component by decryption. The encrypted data will then be transferred to “Image Storage” subsystem for data archiving. “Expert System” subsystem will not interact with “Image Storage” directly, all data access requests will go through “Security” component. As a result, system’s performance will be affected negatively since there are intervention steps and longer data transmission time before diagnosis. Option2 is an alternative solution where images acquired will be stored directly as raw data. “Image Process” and “Expert System” are able to access these data before they are encrypted. “Image Process” only needs to get processing authority through the authentication of “Security”. Obviously system’s performance is improved in Option2, patients are able to get the diagnosis result much faster than Option1, but the system’s security level is reduced to a lower level. This is the sort of information useful in determining the quality of the final product and making tradeoff decisions. Of course, each recommended option also has impact to other quality attributes of the software system and the interaction among the components could be quite complicate, due to the limit of space, we will not present more details in here.

In real world of PACS product line, tradeoff situations are more complicated as more quality attributes have to be considered and even minor quality differences will differ the system components and their connections. By listing the quality related sub-views and combinations, the architectural recommendations are clarified and available for the member product architecture design at the concrete level.

V. CONCLUSION AND FUTURE WORKS

Software architecture design is one of the most critical processes of software development. Non-functional and quality...
requirements are critical factors to the architecture design. Quality based design methods are getting more attentions as they are able to realize stakeholders’ expectations and to guarantee the quality of software products. PLA design is more complicated because of the variabilities existing in SPL. In this paper, we extend QADA method by introducing quality based design methods as extra views (quality view and tradeoff view) and use a case study for illustration. The benefit of this design method is to maintain the requirement traceability and further improves the member product consideration.

Our next step is to focus on the variabilities representations in SPL architecture design. There are already some suggestions for representing the architectural variabilities [11]. We would like to investigate how to incorporate the variabilities in these two additional views we have proposed. Furthermore, we like to investigate how to develop member product architecture from the the reference architecture.

REFERENCES


