Applying Layering Concept to the Software Requirements Analysis and Architectural Design

Yunarso Anang[1,2] and Yoshimichi Watanabe[1]

[1] Interdisciplinary Graduate School of Medicine and Engineering,
University of Yamanashi
4-3-11 Takeda, Kofu, Yamanashi 400-8511, Japan
{g14dma01, nabe}@yamanashi.ac.jp
[2] Department of Computational Statistics, Institute of Statistics
Jl. Otto Iskandardinata No. 64C, Jakarta 13330, Indonesia
anang@stis.ac.id

Abstract. Considering software architecture concurrently and iteratively with software requirements, has been illustrated as a way to increase productivity and stakeholder satisfaction in the twin peaks model software development process. Because this model exposed only *the tip of the iceberg*, and lacks of concrete definitions and techniques, an approach of utilizing this model in the real world has been proposed by applying the concept of the product development process based on Quality Function Deployment. In this paper, we will go further of giving more detail about how to define the requirements along with software architecture. In order to provide a method to define a robust software architecture but to be adaptable to the presence of changing requirements, we apply layering concept to the software requirements analysis and architectural design.

Keywords: requirements analysis, architectural design, layer, volatility, abstraction, twin peaks model, quality function deployment.

1 Introduction

A software development project typically starts with extracting some software requirements from the stakeholders. After that, the developer team uses those software requirements as the input for architectural design to define the architecture of the software system to be built. This process is typical of software development based on the *waterfall* model.

In such development process, the software requirements have become a constraint in the architectural design process. In this process, rather than developing from the scratch, most projects are adopting one of the well known and explicit architectural patterns, or selecting a software architecture by reusing the already trusted components providing the framework for the software application. However, since this framework also prescribe the capability such as the extensibility of the application, if improperly selected in the architectural design, it might be difficult to fix in the latter process. The constraint of architectural design can also impede the changes in the software requirement itself, when the changes cannot be implemented in the already selected architecture. As the result, the stakeholder might not have enough satisfied with the final product.

Software requirements are the important elements to be considered first in software development. Software architecture, which is defined based on software requirements, is also the important thing to be thought in order to fulfill the given requirements at the time the architectural design performed. Software architecture also needs to have the flexibility to deal with future change of software or user requirements. The twin peaks model has been proposed to emphasize that developers should equally give status to the specification of requirements and architectures [7]. Compared to Boehm's *spiral* life cycle model [2], the twin peaks model provides a finer-grain one, a life cycle that acknowledges the need to develop software architectures that are stable, but still adaptable, in the presence of changing requirements. However, though the concept is well explained, there is no detail or concrete explanation of what and how to apply, and, at the time, the software-development community has not yet recognized that such a model represents acceptable practice.

Since the result of requirements analysis affects the overall result of the final product, and it is important to ensure customer will be satisfied with the final product, we will further discuss the method of requirements analysis. In our research, we propose the use of Quality Function Deployment (QFD) as a method to clarify the voices of the customer, and define the product quality as well as the business functions of the product based on them, and take them in the whole development process of the product [6]. An approach of combining the QFD based software requirements analysis with the twin peaks model has been proposed [10]. The approach provides a concrete method of defining and deploying software requirements concurrently with software architecture's components, using the QFD's well-known two-dimensional tables.

In this paper, in order to provide a more detail and concrete method in establishing stable software architecture but to have a range of flexibility over changing requirements, we apply the concept in layered architectural patterns to the software requirements analysis and architectural design. The concept is hereinafter referred to as the layering concept.

The rest of this paper is organized as follows. Section 2 presents the concept in layered architectural pattern. Section 3 discusses how we apply the layering concept in requirements analysis and architectural design. In the section, we also show how we evaluate the approach using an example. Finally, we conclude this paper in Section 4.

2 The Concept in Layered Architectural Patterns

Software architecture is a description of subsystems, components, and relationships among them, required for building a software system. Software architectural design is a process whose purpose is to provide a design for the software that implements and can be verified against the requirements.

Among various numbers of software architectures, those that are found particularly useful for families of systems are often codified into architectural patterns. From several architectural patterns already established [4], we chose the layered architectural pattern, because its layering concept has the benefits to separate functionalities into distinct layers, and it can support flexibility and maintainability if it appropriately defined.

The layering concept helps to structure applications that can be decomposed into groups of subtasks in which each group of subtasks is at a particular level of abstraction. Layers are sorted vertically, where that having the lowest level of abstraction is placed at the bottom, and that on the uppermost level of abstraction is placed on the top. The lower layer has less chance of modifying than those above it. One of software architectural designs applying the layering concept is the principal 3-layer architecture: presentation, domain, and data source [3], as summarized in Fig. 1.

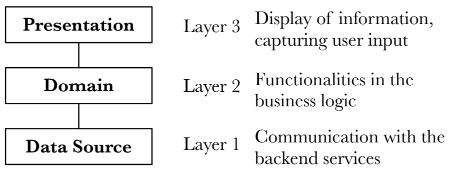

Presentation
Layer 3
Display of information, capturing user input
Domain
Layer 2
Functionalities in the business logic
Data Source
Layer 1
Communication with the backend services

Fig. 1. A diagram showing the principal 3-layer architecture.

The principal 3-layer architecture has a different level of abstraction in each layer. Starting from the bottom, data source layer has the lowest level of abstraction. It means data source should have the smallest potential to change. Changing in data source may have a big impact to the upper layers. In contrary, changes in the application layer, such as changing the color of display or even changing the user interface from desktop to web-based, do not propagate to the lower levels. However, this layer may have a bigger chance of changing compared to lower levels.

3 Applying Layering Concept to the Requirements Analysis and Architectural Design

Software requirements analysis consists of activities including requirements elicitation, requirements analysis, requirements specification, validation, and management activities. The requirements analysis activity is the most important step in the overall development process, as in this activity, the requirements will be classified. The appropriateness of the classification will decide how the requirements would be handled in the architectural design. The software requirements defined in this activity will influence how rigid or stable the software architecture will be designed, and at the same time, that will, indirectly, decide how easy to accommodate in the presence of changing requirements. We need a method of classification which can give an input to the architectural design whose result is adaptable to the future changing of requirements.

In the previous section, we have discussed how the layering concept benefits to increase the extensibility of software development and the flexibility to accommodate changes in requirements. We propose applying the same concept in classifying requirements. We consider the volatility or chance of changing of requirements to be the base of applying the layering concept. There are few researches studying the volatility of software requirements such as in [5][8], but mainly they only study the impact of requirements volatility or its statistics in the development process.

It is stated that some requirements will change during the life cycle of the software, and even during the development process itself. For example, in an online new student admission application, requirements for functions to make the registration online are likely to be less volatile than requirements to support integration of the admission fee payment. The former reflects a fundamental feature in the admission application, while the latter may change as the payment method may vary from manual payment to automatic payment via online banking application which may need a system integration with banking application. It is useful if some estimate of the likelihood that a requirement will change can be made. Flagging potentially volatile requirements can help the software engineer to establish a design that is more tolerated of change.

Fig. 2 shows how we apply the layering concept to the requirements analysis and architectural design. First, the requirements will be sorted based on their degree of volatilities. The sorted requirements then will be used in architectural design, which is in this case, they are mapped into the 3-layer architecture.

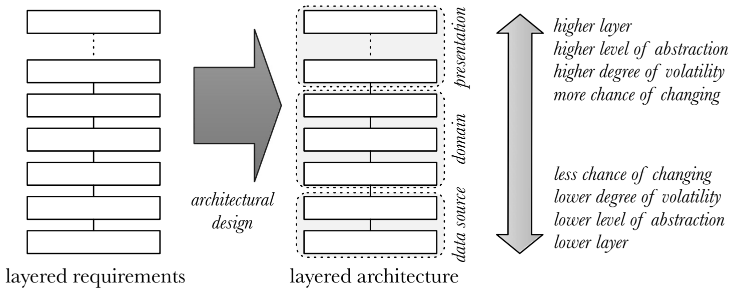
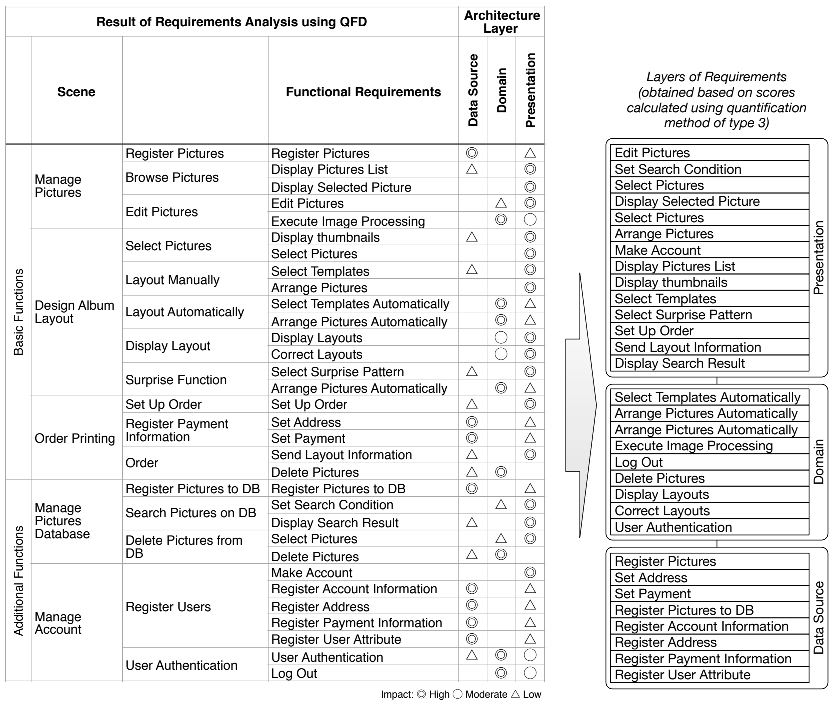


Fig. 2. A process of architectural design from the layered requirements, producing a layered architecture.

To illustrate the approach described above, we evaluate it using an example. The example is taken from the paper introducing the requirements analysis method using QFD [9]. The example is about the virtual software of web based system which provides management of pictures taken while the user traveling.

We use the result of the method described in the paper as the input of our proposed approach as shown in left side of table in Fig. 3.



Result of Requirements Analysis using QFD				Architecture Layer		
	Scene		Functional Requirements	Data Source	Domain	Presentation
Basic Functions	Manage Pictures	Register Pictures	Register Pictures	◎		△
		Browse Pictures	Display Pictures List	△		◎
			Display Selected Picture			◎
		Edit Pictures	Edit Pictures		△	◎
			Execute Image Processing		◎	○
	Design Album Layout	Select Pictures	Display thumbnails	△		◎
			Select Pictures			◎
		Layout Manually	Select Templates	△		◎
			Arrange Pictures			◎
		Layout Automatically	Select Templates Automatically		◎	△
			Arrange Pictures Automatically		◎	△
		Display Layout	Display Layouts		○	◎
			Correct Layouts		○	◎
		Surprise Function	Select Surprise Pattern	△		◎
			Arrange Pictures Automatically		◎	△
	Order Printing	Set Up Order	Set Up Order	△		◎
		Register Payment Information	Set Address	◎		△
			Set Payment	◎		△
		Order	Send Layout Information	△		◎
			Delete Pictures	△	◎	
Additional Functions	Manage Pictures Database	Register Pictures to DB	Register Pictures to DB	◎		△
		Search Pictures on DB	Set Search Condition		△	◎
			Display Search Result	△		◎
		Delete Pictures from DB	Select Pictures		△	◎
			Delete Pictures	△	◎	
	Manage Account	Register Users	Make Account			◎
			Register Account Information	◎		△
			Register Address	◎		△
			Register Payment Information	◎		△
			Register User Attribute	◎		△
		User Authentication	User Authentication	△	◎	○
			Log Out		◎	○

Impact: ◎ High ○ Moderate △ Low

Layers of Requirements (obtained based on scores calculated using quantification method of type 3)

Presentation
Edit Pictures
Set Search Condition
Select Pictures
Display Selected Picture
Select Pictures
Arrange Pictures
Make Account
Display Pictures List
Display thumbnails
Select Templates
Select Surprise Pattern
Set Up Order
Send Layout Information
Display Search Result

Domain
Select Templates Automatically
Arrange Pictures Automatically
Arrange Pictures Automatically
Execute Image Processing
Log Out
Delete Pictures
Display Layouts
Correct Layouts
User Authentication

Data Source
Register Pictures
Set Address
Set Payment
Register Pictures to DB
Register Account Information
Register Address
Register Payment Information
Register User Attribute

Fig. 3. Layering requirements derived from requirements analysis using QFD.

In order to arrange the requirements into layers, we have to obtain the degree of volatility. To obtain the degree of volatility, in this evaluation, we adopt the method of software system's near decomposition which use the quantification method of type 3. This method produces the score of elements from their relations in a two-dimensional table [1]. We treat the scores as scaled values of the degree of volatility. In the middle side of table in Fig. 3, we added 3 columns containing 3 layers taken from the 3-layer architecture. Then we fill in the table how strong the requirement is related to each of the layers. After converting the symbols into scaled numbers, we obtain the requirements-architecture matrix as the input of the quantification method of type 3. The result contains the scores of both requirements and architecture layers. The right side of table in Fig. 3 shows the requirements grouped into 3-layer architecture which are sorted based on the scores. Requirements are arranged from those which are not likely to be changed to those which are having bigger potential of changing. As stated in the twin peaks model, this process should be incrementally conducted in order to obtain finer-grain result.

4 Conclusions

In a standard list of software life cycle processes, such as that in ISO/IEC/IEEE Standard 12207:2008, software design consists of two activities that fit between software requirements analysis and software construction. Nuseibeh has already proposed the twin peaks model by giving the equal status of the specification of requirements and architecture, and to provide more detail and concrete method, Watanabe et al. has proposed to combine the method of software requirements analysis based on QFD with the twin peaks model.

In this paper, we have proposed applying the layering concept to requirements analysis and architectural design, in order to obtain layered requirements and a layered architecture. This layered structure provides a stable architecture but adaptable in the presence of changing requirements.

Although we have provided illustrations for our approach, we exposed only a concept or likely an idea. More work remains to evaluate the validity of the method and the effectiveness of the approach in a real project.

References

1. Anang, Y., Amemiya, A., Yoshikawa, M., Watanabe, Y., Shindo, H.: A Software Tool for Making a Two-way Table Like a Quality Table. In: Proceedings of 11th International Symposium on Quality Function Deployment
2. Boehm, B.: A Spiral Model of Software Development and Enhancement. Computer 21(5), 61–72
3. Brown, K., Craig, G., Hester, G., Pitt, D., Stinehour, R., Weitzel, M., Amsden, J., Jakab, P.M., Berg, D.: Enterprise Java Programming with IBM WebSphere. Addison Wesley (2001)
4. Bushmann, F., Meunier, R., Rohnert, H., Somerlad, P., Stal, M.: Pattern-oriented Software Architecture: A System of Patterns. John Wiley & Sons (2001)
5. Dev, H., Awasthi, R.: A Systematic Study of Requirement Volatility during Software Development Process. International Journal of Computer Science Issues 9(2), 527–533 (2012)
6. Japanese Standards Association: JIS Q 9025:2003 Performance Improvement of Management Systems – Guidelines for Quality Function Deployment
7. Nuseibeh, B.: Weaving the Software Development Process Between Requirements and Architectures. In: Proceedings of 23rd International Conference on Software Engineering, International Workshop on Software Requirements to Architectures
8. Singh, M., Vyas, R.: Requirements Volatility in Software Development Process. International Journal of Soft Computing and Engineering (4), 259–264
9. Watanabe, Y., Kawakami, Y., Iizawa, N.: Software Requirements Analysis Method using QFD. In: Proceedings of 18th International Symposium on Quality Function Deployment
10. Watanabe, Y., Yoshikawa, M., Shindo, H.: Software Development Method based on Twin Peaks Model with QFD. In: Proceedings of 19th International Symposium on Quality Function Deployment