Architecture Decision within Value-Based Software Engineering concepts

Jin-Gyu Kim
Information and Communications University
MunJi-Dong, YouSeong-Ku,
DaeJeon, South Korea
82-042-866-6215
jinlooks@icu.ac.kr

Sungwon Kang
Information and Communications University
MunJi-Dong, YouSeong-Ku,
DaeJeon, South Korea
82-042-866-6215
kangsw@icu.ac.kr

ABSTRACT

Architectures in software intensive systems are a significant field of study while it is representing static structure, dataflow, and relationships among subsystems or components. Also, architectures could be considered an artifact as blueprint of software system to make sure that design approach will yield an acceptable system in early system analysis. As the complexity of system increases, there will be several approaches to design or select components to improve qualities, namely performance, modification, and security, and there are many stakeholders involved in those architectural concerns such as implementors, testers, maintainers, and managers. To select right and workable architectural approaches, architects firstly need to recognize the values among stakeholders, how to affect them, and how to lead negotiated architectural decisions against the value-neutral approach that focuses on only technical issues. In this paper, we primarily propose a method that systematically derives architectural decision framework to reflect both economical and technical issues in context of architecture processes.

Categories and Subject Descriptors

K.6.3 [Management of Computing and Information Systems]: Software Management – *Software Selection, software development, software maintenance.*

General Terms

Management, Measurement, Performance, Design, and Economics.

Keywords

Architecture Decision, Weighed Sum Technique, Business and technical issue, Decision Framework.

1. INTRODUCTION

Software might be the first thing to bridge between customers and products or services, and growing competitive environment in market requires a high-quality, flexible and adaptable software system to support its particular services. To respond to such environment, to understand software architecture and to select suitable software architecture are the most significant steps to develop a software product while architecture is representing static structure, dataflow, and relationships among subsystems or components [1].

There are growing issues in developing software systems such as low productivity, low quality, and high maintenance cost, and many organizations adapt software process, such as CMM, CMMI, Six-Sigma, and ISO, to optimize development and maintenance activities. Even though these methods increase productivity and quality of software systems, it is not the ultimate solution to maximize the values of software systems and to reduce the cost of maintenance without strong architecture reflecting industrial requirement [5]. As the complexity of system increases, there will be several approaches to design or select components to improve qualities, namely performance, modification, and security [1]. Software architects traditionally had an architectural decision process with the fundamental understanding of what software architecture should be, and architects could turn the questions of how to create architecture into good, right and successful technical supports. The primary intent of this paper is to establish a strategy to help understanding the full nature of software architecture process. Then it explores a case study of Message Translation System with specific architectural approaches through value based architecting that involves further trade-offs of the system objectives with achievable architectural solutions [3]. Finally, it suggests a method to generate an architecture decision framework that reflects both economical and technical issues in context of architecture processes.

2. RELATED WORKS

The most commonly accepted and well-known concept about VBSE (Value-Based Software Engineering) is that enabling significant improvements in software design and engineering through economical reasoning about software system is to maximize its benefits in not only present values, but also future values [6]. In comparing past works, we focused on development costs, rather than considering other economical and environmental factors, and we had believed that improving the performance of existing functions or creating high technologies provides enormous benefits. However, recently, we have realized even high technology or new functionality itself does not satisfy objectives of system development until putting economical issues into technical issues. The VBSE concept is very strong at maximizing values of related issues by the principle of separation of concerns, providing practical plan and supported environment, and trade-off analysis among stakeholders [7]. But, we still need to progress on value-based architecting by being absorbed business needs into software architecture.

Another famous trend in software engineering is product line engineering which mainly consists of two parts: domain engineering and application engineering. Its goal is to support the systematic development of a set of similar software systems by understanding and controlling their common and distinguishing characteristics [8]. The methodologies of product line engineering emphasize proactive reuse to construct high-quality products that are less costly but more quickly, so that the productivity increases remarkably and product qualities are guaranteed by maximizing reusability since components in domain artifacts are developed and tested in domain engineering. Also, product architecture freely derived from artifacts of domain engineering with variability points for reflecting specific product requirements. However, variability points in domain artifacts only allow technical flexibility; moreover, whole product line approach is itself limited within reducing development costs and time to market. It is necessary to place VBSE concepts in designing software architecture with the following questions.

- How can one provide architectural solutions combining technical and business issues?
- How can one trade-off or reflect software architecture decisions between business needs and technical supports?

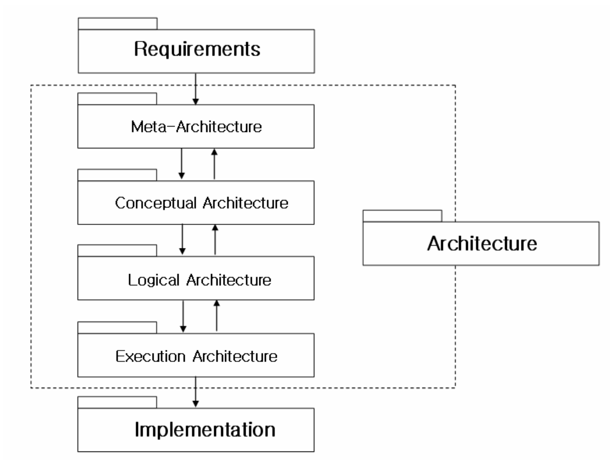

Figure 1. Architectural Processes between requirement and implementation

3. ARCHITECTURE PROCESSES

After requirement engineering, essentially, there are architectural activities to provide structural information to software developers. Architecture is simply to figure out what components are needed and what relationship are required among them; however, there are many subprocesses within the architecture phase, namely, meta-architecture, conceptual architecture, logical architecture, and execution architecture. Those subprocesses are composed as a layered structure, and they conduct iteratively (as figure 1) until all the architectural requirements are satisfied.

3.1 Meta-Architecture

Any decisions in the meta-Architecture strongly affect the integrity and structure of the system; however, it does not itself define the structure of the system [9]. The meta-architecture provides us with the limitations of or recommendation for architecture style, principles, philosophy, and pattern of composition or interaction by architects through extracting and analyzing the architectural requirements from requirement phase. This phase is very helpful rather than directly moving on the conceptual architecture phase in that architects think about the qualities that system should delivered and the components needed in conceptual architecture.

3.2 Conceptual Architecture

The main activities in the conceptual architecture are to identify the high-level components of the system and those relationships among one another [9]. In this phase, architects can directly focus on a suitable decomposition rules of the system without considering detail designs. Moreover, artifacts in conceptual architecture can be communication tools between architects and non-technical audiences, such as managers and users, through architecture diagrams and informal component specifications [4].

3.3 Logical Architecture

The output of the logical architecture is “Blueprint” of whole systems with precise and unambiguous properties. At this phase, all artifacts from the conceptual architecture became visible and are assembled with well-defined interfaces and component specifications. Key architectural mechanisms also supported onto or among the components [9]. By doing those activities, each component is developed individually and located in architecture diagrams with enough explanations and rationales.

3.4 Execution Architecture

Execution architecture is for describing hardware concerns of software systems, such as throughput and scalability. In distributed and concurrent mechanisms, execution architectures; for instance, development and deployment views, are the suitable tools to show the mapping of components onto nodes, or mapping of components onto processes of the physical systems [9].

Significant architectural-decision processes occur between meta-architecture and conceptual architecture, or between conceptual architecture and logical architecture [9]. Architects may consider lower-level designs or implementations in architecture decisions; however, they should not be regard as architecture ones since it has only local impact, not systematic impact, and it does not require making the necessary trade-offs across the system [9]. Indeed, Architectural-decision processes should be included a broad-scoped or system perspective. A case study of Message Translation System in the next chapter offers visual understandings of architectural decisions in more detail.

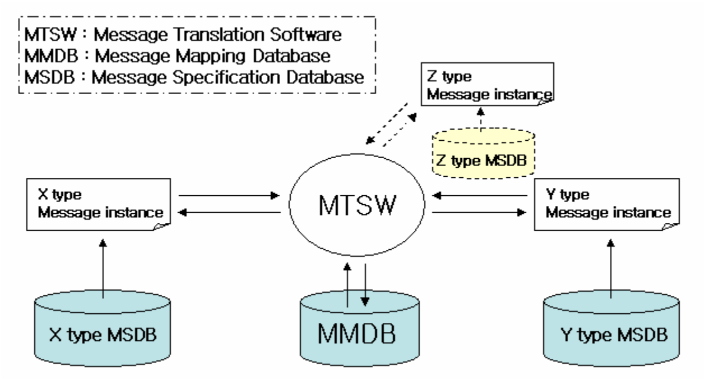

Figure 2. The current structure of Message Translation System with an extension plan

4. CASE STUDY: Message Translation System

The organization that developed the Message Translation system considers extending the current system by adding more message type. The current system, which has only two types of message translating, consists of MTSW (Message Translation Software) and its MMDB (Message Mapping Database), as shown in Figure 2. A message instance is generated from its MMDB, and it goes into MTSW. Based on message element mapping tables in MMDB, the message instance is transformed into different type of message elements and reconstructed those message elements according to the syntax rules. However, with three more message types, the current system structure is not effective, and it requires new architectural approaches since one-by-one mapping rule does not work among different types of elements. Theoretically, it requires one-to-many or many-to-many mapping rules; that is one element or a group of elements transformed as one or a set of elements in a different type of message.

It will take less than six months for this organization to develop the extension of this system with designing components related new type of messages due to the business needs. When a message instance is translated into a different type, it should keep the same translation time as the current system dose. Additionally, the components of the current system should be reused in new extended system.

4.1 Three architectural options

To address this challenge and meet the emerging business needs, the organization developed architectural visions to guide its system's transformation through numerous infrastructure improvement studies and they finally narrow down their visions to three workable options (see Figure 3).

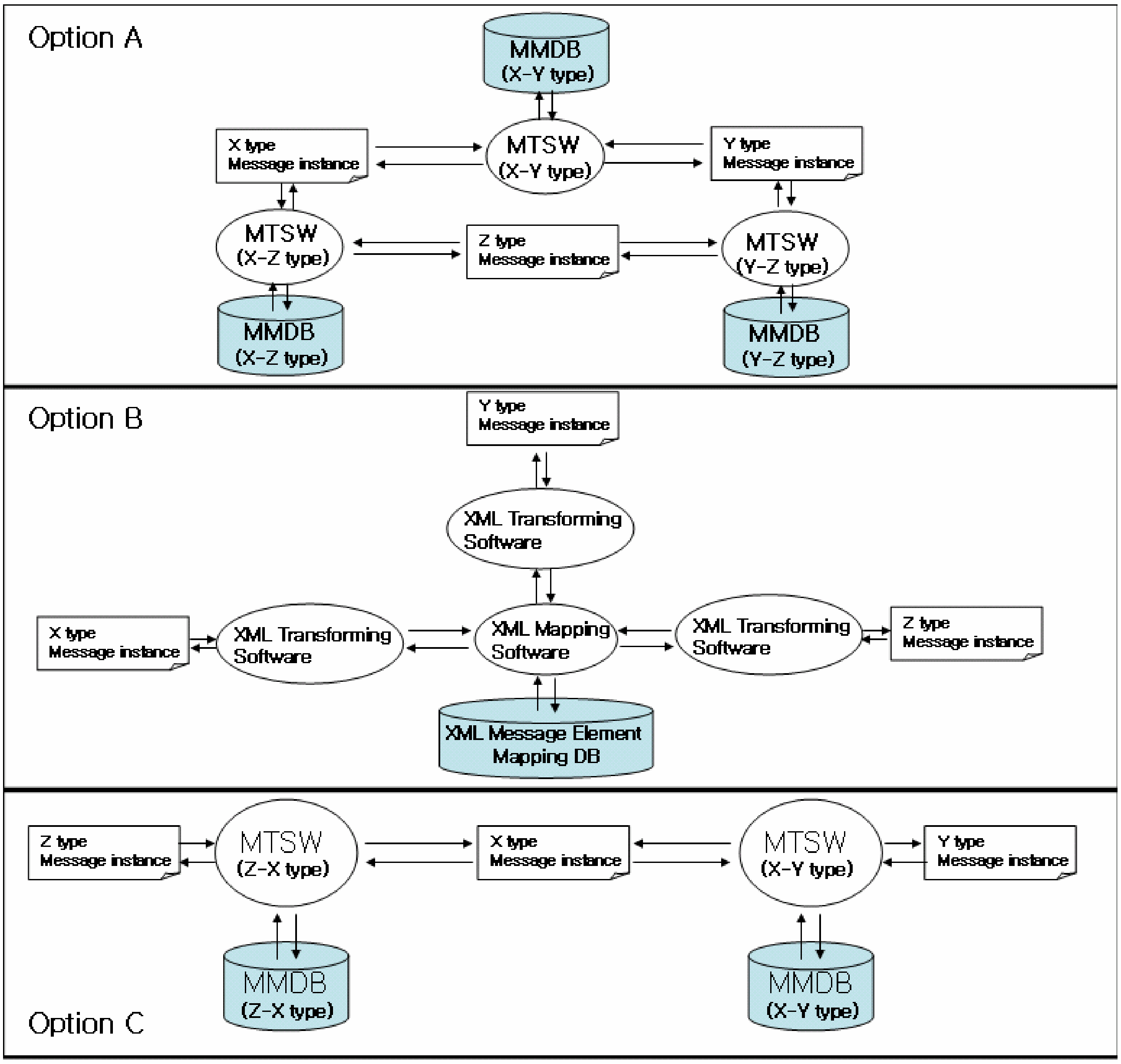

Figure 3 Architectural approaches

Option A. In this approach, when a new message type appears and is added to Message Translation System, it lets each message type have a one-by-one relationship. For example, suppose Z type of message is added to this system among X and Y types of messages, it requires additional message translation software program with the message mapping database between X and Y messages. Also, another message translation software program and its message mapping database located between Y and Z messages. This approach does not require further technologies or COTS (Commercial off-the-shelf) from the outside, and the current components are fully reused; however, when a new type of message added, more than two message translation software program and message mapping databases are needed so that the cost and time of development components and complexity of structure rapidly increase.

Option B. This XML (Extensible Markup Language) intensive translation approach is all messages firstly transformed into the XML syntax structure and are sent to the XML mapping software. This software program is mapping and exchanging other kinds of XML message elements through accessing XML message element mapping database and reconstructs the XML syntax structure in different message types. With this approach, whenever a new message type is developed, the message is easily added to this architecture by transforming all the message elements into the XML structure and updating them onto the XML message elements mapping database. This approach has strength of reducing development cost and time in further extension, but it needs transforming the current structure of system. Additionally, it requires further technology such as XML transforming; however, this technical issue can be solved by COTS, which fully supports XML transforming.

Option C. The idea of this architectural approach is to select one of the message types and use it as a bridge for other message types; for instance, if X type of message is selected as a bridge message, this message is located in the middle of the system and message translation software and its message mapping databases are positioned between a bridge message and other types of messages. This approach mainly contributes to saving the development cost and time and keeping the original structure to reduce the complexity of system. Moreover, it is free from the issues of further technology. However, when an instance message translates into a different type of message, it demands translation twice in this architecture except for the translation to the bridge message, and this means the translation time is much longer than the current system's.

Table 1. Architecture Decision Table of Message Translation System

		Option A		Option B		Option C	
Essential Criteria							
- Be ready for delivery in Six months?		Yes		Yes		Yes	
- Keep original translation time?		Yes		Yes		No	
- Reuse original components?		Yes		Yes		Yes	
Selected Criteria							
	Weight	Option A		Option B		Option C	
1. Development Costs?	6	-1	-6	0	0	1	6
2. Reduce time to market for future enhancement	5	-1	-5	1	5	0	0
3. Require further technology?	4	1	4	-1	-4	1	4
4. Available outside COTS?	3	-1	-3	1	3	-1	-3
5. Transforming original structure?	2	1	2	-1	-2	1	2
6. Development Time?	1	-1	-1	0	0	1	1
Total	21		-9		2		10

5. DECISION FRAMEWORK

This Architectural Decision Table basically consists of two parts; essential criteria and selected criteria, as described in Table 1. First of all, architects are provided the list of criteria from system analysts or themselves extract them from meta-architecture phase. Then they classify this list into the essential list and the optional list. The essential criteria indicate the lists which the alternatives should be satisfied; however, the alternatives in selected criteria part would be considered as optional supports. Those selected criteria provide or lead architects to which alternative is the most suitable for specific situation or development team. On the other hand, within essential criteria area, architects simply ask whether certain architectural approach supports it or not. It means they do not take any rating techniques into consideration on the essential criteria, and some alternatives would be eliminated from the Architecture Decision table in case that those were not met with a certain criterion.

The selected criteria part is located in bottom of Architectural Decision Table and is applied by weighted sum technique. This part consists of optional criteria and those are already prioritized with a set of weights to reflect the relative significant of each criterion. Moreover, those criteria are assigned to a rating for each criterion to reflect how well options satisfy the criterion. As the result of this analysis, architects consequently recognize which alternative is the most suitable.

5.1 Traditional problems with Weighted Sum Technique

Problems with weighted sum technique are that wrong set of rating factors and weights suggested unacceptable alternatives. For instance, some criteria require critical considerations and they should be classified in high-level criteria. However, it is hard to figure out accurate ratings and weights in early design phase [2]. In the Architectural Decision Table, those requiring critical consideration or architect seems it is significant could be tie up together in the essential part to justify critical decision first.

Another issue in weight sum technique is how well assigns or fairly contribute ratings and weights to each criterion according to their significations [2]. For more specific, one criterion possibly makes other criteria meaningless; for example, suppose development costs (see table. 1) was assigned considerably high weight with high rating relatively to the other criteria, the other criteria could not have any effect on this architectural decision. To minimize this problem, the Architecture Framework Decision is discriminated just by one between criteria; so that it gives the chances that lower level criteria can overcome or lead different alternatives even though a higher level criterion supports a certain alternative while most critical criteria are classified into the essential part. The formula of assigning weights is like below.

$N-(L-1)$ ……………………………………...…. **[Eqn, 1]**

$\sum_{L=1}^{N} W_L$ ………………………………..…………. **[Eqn. 2]**

W_L : Weighted value of certain level
N : Number of selected criteria
L : Level of each selected criterion

5.2 Applying modified Weighted Sum Technique

Firstly, architects prepare levels of criteria categories as same size as the number of the selected criteria. They prioritize criteria and assign weights to each criterion using Eqn. 1. After that, they position the criteria into the selected criteria part from most significant one with a certain weight. Regarding the rating table (see Table. 2), it simply divide into three categories such as ‘Strong’, ‘Acceptable’, and ‘Poor’, and it just uses 1, 0, and -1 for allocated value to minimize differentiation of weighted rating value between category levels. For example, if alternatives are strong at certain criterion, it will have plus effect based on its weight, and if its rating is acceptable, it will have no effect on it; otherwise, it will have minus effects in case of poor at this criterion. After filling out rest of all Architecture Decision Table through multiplying weight by its rating value, each weighted rating is sum up to compare which alternative is the most suitable. Eqn. 2 is used to figure out sum of weights in Architectural Decision Table.

Table 2. Rating Table

Rate	**Value**
Strong	1
Acceptable	0
Poor	-1

Regarding the result of Table 1, **Option C** should be considered as the most alternative within the selected criteria section; however, **Option C** does not suitable while it does not support or satisfy the translation time as same as the current system’s, and it requires two times of translation when Y type of instance message is translated into Z type of instance message or in reverse. So, **Option B**, which is second ranking in selected criteria, will be selected as the further development type of structure.

6. CONCLUSION

Architecture in software intensive system is the central concern to ensure the qualities or properties of system and its business objectives. However, what seems to be lacking is to combine the concept of technical issues and business issues into the software architecture. During the software architecture designing, architects had just focused on the qualities of system such as performance, modification, maintainability, and security across components or subsystem, and they simply believed that the high quality software system satisfies its business needs. Therefore, this paper have considered fundamental steps of architectural process and applied VBSE concepts to bring business needs into architectural decisions. By doing this, business issues, occurred from outside of development team, embodied into technical software architecture along with architectural process. Furthermore, it proposes and illustrates the usefulness of the method by conducting a case study of Message Translation System and proved how uncertain software architecture flows out and refines into the good, right, and successful architectural solution by driving modified weighted sum analysis technique.

Future researches, proposed in this paper, include the development of domain architecture with value maximizing methods in software product line engineering, especially with variability management. In particular, to define technical and business issues at same variation points will be inevitably an essential activity for improving the speed of developing high-quality products and reducing those costs. This would be more the efficient management in domain level of product line architectures.

6. References

[1] Len Bass, Paul Clements, Rick Kazman: *Software Architecture in Practice*: Addison Wesley 2003.

[2] Barry W. Boehm: *Software Engineering Economics*: Prentice-Hall 1981.

[3] Stefam Biffl, Aybuke Aurum, Berrry Boehm, Hakan Erdogmus, Paul Grunbacher: *Value-based Software Engineering*: Springer 2006.

[4] Ruth Malan, Dana Bredemeyer: *Conceptual Architecture Action Guide*: Bredemeyer Consulting.

[5] Jongsu Bae, Songwon Kang: *A methods to generate Feature Model from Business Process Model for Business Application*: Information and Communications University 2007.

[6] Barry W. Boehm, Kevin J Sullivan: *Software Economics: A Roadmap*: University of Southern California, University of Virginia Thornton Hall.

[7] Steve McConnell: *Rapid Development*: Mircosoft Press 1996

[8] Klaus Pohl, Günter Böckle, Frank J. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*; 1st ed, Springer, September 19, 2005.

[9] Ruth Malan, Dana Bredemeyer: *Software Architecture: Central Concerns, Key Decisions*: Bredemeyer Consulting.