Making Architectural Decisions Based on Requirements: Analysis and Combination of Risk-Based and Quality Attribute-Based Methods

Bingfeng Xu Zhiqiu Huang Ou Wei
College of Information Science and Technology
Nanjing University of Aeronautics and Astronautics
Nanjing, P.R.China, 210016
{xubingfeng, zqhuang, owei}@nuaa.edu.cn

***Abstract*—Software architecture is critical to the success of large software systems. It has long been recognized that architectural design has profound impact on the achievement of system requirements. Two typical methods have been proposed to help users to make architectural decisions based on requirements. One method uses risk-based reasoning to help choosing good architectural design that can meet both system requirements and resource limitations; the other one, using quality attribute model as the linkage, enables deriving a fragment of architectural design that is focused on achieving certain quality requirement. However, there is little effort on comparing the two methods to discover similarities and differences. In this paper, we conduct a systematic analysis of the two methods, and compare them from the aspects of system requirements, architectural decisions, and mapping approach. We then propose a procedure to combine the two methods that provides better support to architectural design decision making.**

Keywords-Requirements; Software Architecture; Decision Making; Risk-Based Reasoning; Quality Attribute-Based Method;

I. INTRODUCTION

Both requirements engineering and software architecture are important areas of software research. Requirements engineering focuses on the problem domain and the responsibilities of software systems. The goal of requirements engineering is to obtain clear understanding of the problem domain and customer's need, and produce complete and correct software requirements. Software requirements describe what a system should achieve, the contexts where it will be used, and the constraint of the functions provided by the system. Software architecture involves the study of the properties and interactions of system elements, and the patterns of composition and constraints on these patterns. Software architecture is a key part to support system engineers, software developers and customers to communicate and understand the system being built [1]. Over the past 20 years software architecture has received increasing attention as an important subfield of software engineering. Researchers and practitioners have realized that choosing an effective architecture is a critical factor for system design and development [2, 3]. And then it has been recognized that quality and attributes (such as maintainability, reliability, usability, performance, and flexibility) of large software systems are largely constrained by the system's software architecture [4].

It is generally accepted that there is close relationships between software requirements and architectural designs. Architectural design decisions underlie the software architecture. As such, software architectures can be seen as the result of a set of architectural design decisions [5]. In order to build a high quality software system, the software designers should be able to analyze and evaluate the effects of various design decisions on the achievement of system requirements at an early state [6]. A lot of effort has been put into the research to bridge the gap between software requirements and architectural designs [7, 8]. In this paper, we study two methods that help software designers to make architectural design decisions based on requirements. One method uses risk as guide to help choosing the architectural style which can match system requirements and meet resource limitations [9, 10]. The other one, using quality attribute model as the linkage, enables moving from quality attribute requirements, through architectural tactics, to design fragments that realize the quality goals [11, 12]. We conduct a systematic analysis of the two methods from the aspects of system requirements, architectural decisions, and mapping approaches. Our analysis shows that the two methods are not comparable, but complement each other: while the risk-based method is more appropriate for architectural style analysis based on generally specified system requirements, the quality attribute based method is good for detailed architectural decision making. To provide better support to architectural design decision making, we then proposed a procedure to combine both risk-based and quality attribute-based reasoning.

The rest of the paper is organized as following: Section 2 introduces the risk-based method, and section 3 introduces the quality attribute based method. We compare the two methods in section 4. We then propose a procedure to combine them in Section 5. We conclude the paper and discuss future work in Section 6.

II. RISK-BASED METHOD

Risk-based method is based on "Defect Detection and Prevention (DDP) process" [13, 14], a risk management framework developed and used at JPL to help engineer to make choices in designing spacecraft and associated technology [15]. A quantitative model in DDP is used to provide a foundation for reasoning and help to optimize early decision making for

978-0-7695-4272-0/10 $26.00 © 2010 IEEE
DOI 10.1109/UIC-ATC.2010.74

maximizing requirements attainment. Having observed that the standard DDP process is incapable to explicitly capture the effect of software architectural decisions on the risks associated with requirements, Kiper and Feather included architecture as a first class concept and proposed a revised DDP process [9, 10]. They claimed that the revised process is able to lead to good choices of software architectures that can achieve the system requirements and be implemented within budget and resource constraints. In the rest of this section, we first introduce the key concepts in DDP process, and then describe the revised DDP process.

A. Key Concepts

The DDP process involves the following key concepts that it captures and tracks [5]:

- **Requirements** in DDP process contain the information about what a system is to achieve, and constraints of the operation environment (such as schedule and budget information).
- **Risks** are all the things that would prevent the fulfillment of certain requirements if they occur (such as uncertainty, flexibility and fault tolerance).
- **Mitigations** are the activities to reduce the chances of risk occurring and/or reduce their impact on requirements.
- **Architectures** refer to the classic software architectural styles in DDP process, e.g. blackboard, pipe and filters, abstract data type, or the hybrid architectures designed by software engineers.

In the standard DDP process, requirements are related to risks, and risks are related to mitigations. The scores assigned for the relations between requirements and risks, and the ones between risks and mitigations provide a basis for risk-based quantitative requirements reasoning, which is used to help choosing cost-effective mitigations.

Since software architecture may be good for reducing certain system risks and then increase the level of requirement attainment, architectural decisions are encoded as mitigation choices in the standard DDP process. Some issues, however, may be caused by this. For example, new risks could be induced by the choices of software architectural designs, and the existing risk may be aggravated because of this. Furthermore, the risks and mitigations derived from an architectural choice in turn can influence how well that architecture mitigates the original risks. To address these issues, a revised DDP process is proposed by Kiper and Feather.

B. Revised DDP Process

The revised DDP process is shown in Figure 1 [9,10], which includes two phases:

- The first phase deals with requirements, risks and architectures. In this phase, the alternative software architectures used to reduce system risks are explicitly captured, and a tentative architecture is chosen.
- The second phase focuses on architectures, risks and mitigations. The tentative architecture passed from the first phase is considered as requirements in this phase. Mitigations then are analyzed based on how they can reduce the risks that may prevent the attainment of the architectural requirements.

At the start of the process, requirements are collected and assigned weights by engineers and domain experts. Engineers and experts then list the possible software architectures to reduce the risks. They also need score each architecture according to its ability to reduce the risks, and save the values in Risk-Architecture matrix. Based on this information, designers can select a software architecture as tentative choice.

The tentative architecture initiates the second phase of the DDP process and serves as requirements in this phase. Similarly to the first phase, the list of risks are identified. After the information of risks is obtained, mitigations that can reduce the impact of the risks are identified. To build a quantitative linkage between risks and mitigations, engineers and experts need score each mitigation against each risk based on its effect on reducing the risk. These values are saved in Risk-Mitigation matrix. Combining the information between risks and architectures, and the ones between risks and mitigations, designers now are able to decide what mitigation activities can best achieve the system requirements using the tentative architecture, while satisfying budget and resource constraints.

The entire DDP process may be iterated to capture the requirements and additional risk that are discovered later by designers.

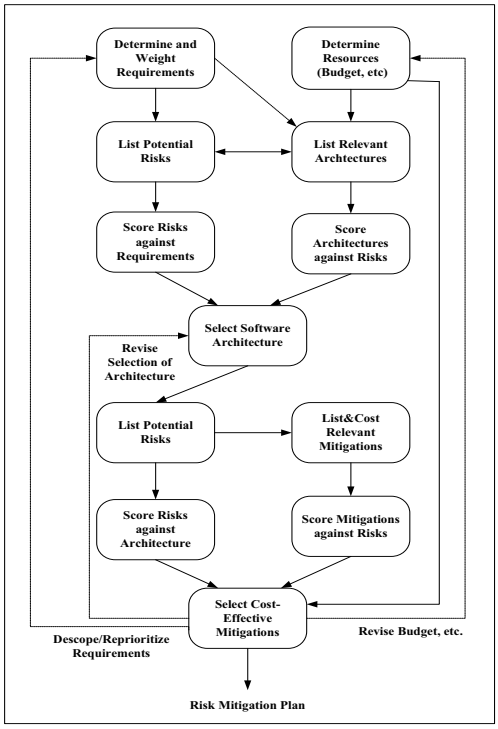

Figure 1. Revised DDP process.

III. QUALITY ATTRIBUTE-BASED METHOD

The key principle of quality attribute-based method is that the quality attribute requirements (such as performance, modifiability, reliability, and usability) exert a dominant influence on the decisions of software architectural designs [16]. In this method, architectural tactic is introduced as a notion to codify these influences, and an architectural design fragment provide solutions to achieving the requirement. Quality attribute model is used as the linkage between specification of a quality attribute requirement and an architectural design fragment. A series of analysis-based steps are then proposed in the method to guide users to derive a set of architectural tactics with respect to achieving a particular quality attribute requirement, and then use the results to make design decisions [11, 12].

A. Key Concepts

We first introduce the key concepts used in the quality attribute-based method.

- **Quality Attribute Scenarios** There are two kinds of quality attribute scenarios, general scenario and concrete scenario. A general scenario is used to specify system-independent quality attribute requirements. To specify the requirement for a particular system, general scenarios are converted to concrete scenarios by replacing generic vocabularies with concrete, system-specific vocabularies.
- **Architectural Tactics** The architectural decisions that are used to achieve a desired quality attribute requirement are characterized as architectural tactics. Using architectural tactics, users can make architectural design decisions to satisfy a quality attribute requirement by adjusting some aspects of a quality analysis model.
- **Quality Attribute Models** A quality attribute model is an instance of a quality attribute reasoning framework, which links the quality attribute requirements specified in scenarios and architectural design decisions encoded by architectural tactics.

B. Analysis Steps

The quality attribute-based method defines a process of moving from quality requirements specified by concrete scenarios, through tactics, to design fragments that satisfy the requirement, which is illustrated in Figure 2 [12]. This process includes the following steps:

- Pick a concrete scenario.
- Type-check concrete scenarios.
- Identify candidate reasoning frameworks.
- Determine the bound and free parameters.
- Determine the tactics associated with the free parameters.
- Estimate an initial set of values for free parameters.
- Use tactics to develop satisfactory bindings for all free parameters.
- Allocate responsibilities to the architectural elements of the design fragments that are associated with the selected tactics.

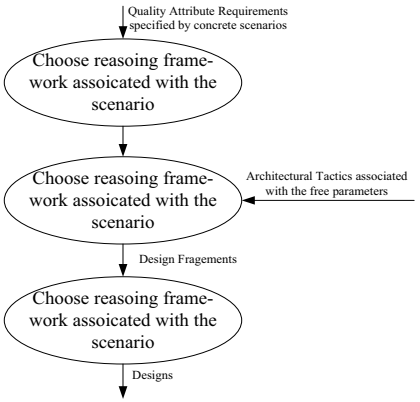

Figure 2. General process of quality attribute-based method.

IV. ANALYSIS OF RISK-BASED AND QUALITY ATTRIBUTE-BASED METHODS

We have introduced the risk-based and quality attribute-based methods for making architectural decisions based on requirements. In this, we compare the two methods in details. We notice that both methods start from requirements of a software system, define a mapping from requirements to architectural designs, and then based on the mapping approach to make decisions of software architectural designs which can satisfy the requirements. Based on this general procedure, we compare the two methods from the following aspects: system requirements, architectural decisions, and mapping approach.

A. System Requirements

In the two methods, different kinds of requirements are used to drive the whole decision making process. In the quality attribute-based method, the requirements sent to the process are quality attribute requirements that precisely specified by quality attribute scenarios. These scenarios describe how and where a quality attribute originates, which parts of a system are affected by this attribute, how the system responses, and how to evaluate the response. These detailed information will be used later to set parameters in reasoning frameworks. The specification of each quality attribute requirement must be checked against the possible values, which have been defined in general-scenario-generation-tables. This can help users to specify quality attribute requirements correctly and completely. However, it may limit the applicability of this method to complex situations where new quality attribute that have not been defined in general-scenario-generation-tables are found and need to be analyzed.

In contrast, requirements used in the risk-based method have no such constraints, which generally describe whatever

the system should achieve. Correspondingly, the risks associated with the requirements do not have information about how to precisely evaluate their effects on requirements. For example, the risk of "algorithm change" just indicates a possible event that may affect the system performance in future. This is a characteristic induced by DDP process, since DDP process is originally designed for decision making in early software development phase where detailed information are always incomplete and uncertain.

Actually, in DDP process, we think that it is both requirements and risk that are driving the architectural decision making process. Compared with quality attribute scenarios, requirements and risks contain the general information of how a quality problem is generated, but do not have information about how to precisely measure the quality. This provides the risk-based method a kind of flexibility for requirements reasoning in situations where detailed information is unavailable, but limits its ability for in-depth analysis of relationships between requirements and architectural designs.

Risks and associated requirements in DDP process can be thought of as descriptions of a system's non-functional properties. Quality attribute requirements (such as performance, modifiability, and reliability) are also concerned with non-functional aspects of a system. This is consistent with the observation that it is the non-functional requirements that motivate the software architectural design [17].

B. *Architectural Decisions*

The two methods produce different kinds of architectural design decisions at the end of the analysis process. Risk-based method focuses on the system-level use of architecture. A collection of possible software architectures styles or hybrid designs for the system being developed are provided by software engineers. Based on risks and mitigations analysis, one of them is chosen as the suitable architecture when DDP process is finished. A software architectural style describes the components and connector types and the constraints of data and control interaction among them. It also informally describes the benefits and drawbacks of the style [18]. Using architectures is very helpful, because it allows the designers to take advantage of experience-based knowledge of many preceding designers who once faced the similar problems.

However, viewing systems at the architectural style levels can only help users to understand the essential behaviors and interactions of the components and connectors. Even if two systems have the same architectural style, in order to satisfy different specific quality requirements, the architectural elements in the systems could be different, and even the elements playing the same role in the architectural style may have different architectural properties and responsibilities.

Unlike the risk-based method, the quality attribute-based method focuses on the derivation of a set of relevant architectural tactics that are further used to determine suitable design fragments to realize quality goals. This allows for architectural decisions to be made in a finer grained way. Furthermore, architectural tactics are codified as design knowledge that is concerned with the relationship between design decision and quality attribute response. Architectural tactics are the building blocks of architectural styles and the carriers of quality attribute-based design knowledge. An architectural style, which is the carrier of system level design knowledge, is in turn the application of one or more tactics.

As discussed before, the difference between architectural decisions used in the two methods is determined by the associated requirements. We have seen that in DDP process, risks and the related requirements only provide general description of quality requirements that need to be attained. Such information may enable designers to propose different architectural styles based on the high-level understanding of the system, but it is not precise enough for analyzing lower level architectural designs such as architectural tactics. Currently, one quality attribute requirement is linked to one design fragment through architectural tactics. According to [12], how to compose the design fragments into a design is still an open issue for quality attribute method, and it should be resolved for this method to be successful.

C. *Mapping Approaches*

The two methods adopt different mapping approaches that are used to link requirements properties and architectural decisions and analyze the linkage relations. In the quality attribute-based method, quality attribute reasoning frameworks link the quality attribute requirements specified in scenarios, and architectural design decisions codified as architectural tactics. Specific reasoning frameworks have been defined for each quality attribute (such as queuing theory and scheduling theory for performance attribute). Each reasoning framework has a set of parameters. During the decision procedure, with the aid of a set of rules, users can adjust the values of the parameters using related architectural tactics, and then identify design decisions for achieving quality attribute requirements. Therefore, reasoning frameworks provide detailed and formal explanation of how different design decisions affect the quality goals.

In contrast, the analysis and reasoning procedure in the risk-based method is not as formal as that in the quality attribute based method. The effect of each architectural style on reducing risks is used as the linkage between architectural styles and the system properties expressed by requirements and associated risks. However, instead of being guided by certain well-established theories, the reasoning of the effects is generally conducted based on experience. After requirements, risks, and possible architectural styles are identified, engineers and domain experts estimate the values saved in Risk-Architecture matrix mainly based on their previous experience. These values then become the basis of the quantitative analysis (e.g. adding up the cumulative impact of risks). As a result, the success of this method largely depends on the experience of engineers and domain experts.

An advantage of the risk-based method is that requirements of the development process (such as schedule and budget profile) are also considered in decision making. This project-related information is represented as migrations and analyzed using Risk-Mitigation matrix. This is helpful for choosing the architecture style that not only satisfies system requirements but also can be implemented within the resources constraints.

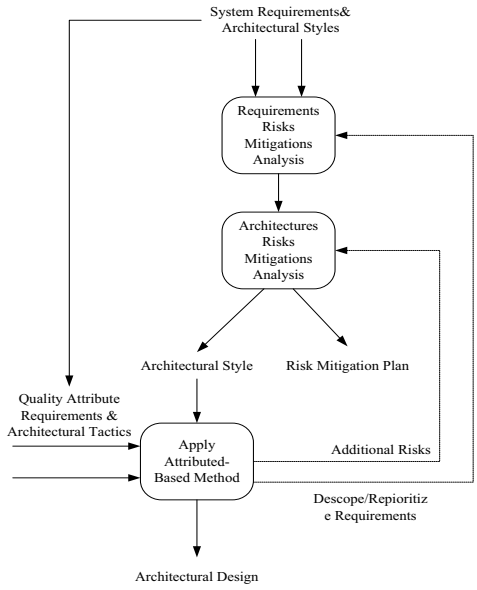

Figure 3. A procedure that combines risk-based and quality attribute-based methods.

V. COMBINATION OF RISK-BASED AND QUALITY ATTRIBUTE-BASED METHODS

Based on the above analysis, we show that the risk-based and quality attribute-based methods are not comparable, but complement each other: while risk-based method is appropriate for architectural style analysis based on generally specified system requirements, the quality attribute-based method is good for detailed architectural decision making using quality attribute requirements precisely specified by scenarios. Based on this analysis result, we propose a requirements-based architectural design procedure that takes advantage of the strengths of both methods.

The procedure is illustrated in Figure 3. The difference between this procedure and DDP process is that we add the quality attribute based analysis in the new procedure. After an architectural style is chosen, we refine original requirements and the associated risks into precisely specified quality attribute requirements. We then apply design fragments derived from quality attribute-based analysis to the chosen style to get more detailed architectural design. It is possible that designer may discover additional requirements and risks in detailed architectural design phase. These information will be sent to previous phases for analysis. Therefore similar to the original DDP process, the whole procedure is also an iterative procedure.

Because the architectural style that passed from DDP process is expected to meet both the original system requirements and resource constraints, it provides a reasonable starting point for detailed architectural design. Furthermore, the ability of performing the detailed design provides a better way to achieve the original architectural style. This may also lead to more effective mitigation of the risks that motivate the selection of the style in the requirements-risks-architecture analysis phase [9]. Applying quality attribute-based analysis to the chosen architectural style can be considered as a way to fine-tune the original style, and thus, reinforces its ability to satisfy requirements. In addition, the formal analysis based on reasoning frameworks can help designer to obtain a deeper understanding of the elements of the architectural style and their interactions, and verify the experience-based analysis conducted in DDP process.

VI. CONCLUSION AND FUTURE WORK

In this paper, we have analyzed two requirements-based architectural decision making methods. One method uses risks as guide to help software engineers to choose the architectural styles that match system requirements and meet resource limitations; the other one, using quality attribute model as the linkage, moves from quality attribute requirements, through architectural tactics, to design fragments that satisfy the quality goals. The main contribution of this paper is that we have conducted a systematic analysis of the two methods from the aspects of system requirements, architectural decisions, and mapping approach. Furthermore, to take advantage of the strength of the two methods, we have also proposed a procedure to combine them.

Our analysis of the two methods shows that due to the complexity of the development of software systems, making architectural design decisions based on requirements is a complex activity. To develop high quality software systems, we need analyze relationships between requirements properties and architectural designs from various aspects, and use information with different levels of detail. As discussed in this paper, risk-based and quality attribute-based methods complement each other. We believe that combination of the two methods is helpful for designing good software architectures. In future, we would like to evaluate the combination procedure we have proposed in the paper. We plan to do the evaluation by conducting case studies, and apply the method to industrial applications.

ACKNOWLEDGMENT

This work is supported by the National High-Tech R&D Program of China (863 Program) (No.2009AA010307).

REFERENCES

[1] M. Shaw, D. Garlan, “Software Architecture: Perspectives on an Emerging Discipline,” Prentice-Hall, Upper Saddle River , 1996.

[2] P. Clements, R. Kazman, M. Klein, “Evaluating Software Architectures: Methods and Case Studies,” Addison-Wesley , 2002.

[3] C. H. Lung, K. Kalaichelvan, “An Approach to Quantitative Software Architecture Sensitivity Analysis,” International Journal of Software Engineering and Knowledge Engineering, vol. 10, No. 1, pp. 97-114 2000.

[4] P. Bengtsson, “Towards Maintainability Metrics on Software Architecture: An Adaptation of Object-Oriented Metrics,” First Nordic Workshop on Software Architecture, Ronneby, 1998.

[5] A. Jansen, J. Bosch, P. Avgeriou, “Documenting after the fact: Recovering architectural design decisions,” Journal of Systems and Software 81(4): 536-557 , 2008.

[6] L. Chung, B. Nixon, E. Yu, "Using non-functional requirements to systematically select among alternatives in architectural design," In

[7] J. C. Duenas, R. Capilla, "The decision View of Software Architecture," EWSA, pp. 222-230, 2005.

[8] J. Bosch, "Software Architecture: The Next Step," EWSA, pp. 194-199 , 2004.

[9] J. D. Kiper, M. S. Feather, "From requirements through risks to software architecture for plan-based and agile processes," In Proceedings of the Workshop on Requirments Engineering for Adaptive Architectures, Monterey Bay, CA , 2003.

[10] J. D. Kiper, M. S. Feather, "Requirements, architectures and risks," In The Second International Workshop on From Software Requirements to Architectures (STRAW'02), Portland, OR , 2003.

[11] F. Bachmann, L. Bass, M. Klein, "Deriving architectural tactics - a step toward methodical architectural design," Technical Report CMU/SEI-2003-TR-004, Software Engineering Institute, Carnegie Mellon University, 2003.

[12] F. Bachmann, L. Bass, M. Klein, "Moving from quality attribute requirments to architectural decisions," In The Second International Workshop on From Software Requirements to Architectures (STRAW'02), Portland, OR, 2003.

Proceedings of The First International Workshop on Architectures for Software Systems, Seattle, WA , 1995.

[13] M. S. Feather, "Defect Detection and Prevention(DDP)," Monterey Workshop pp. 13-14, 2007.

[14] S. L. Cornford, "Managing Risk as a Resource Using the Defect Detection and Prevention Process," In: 4th International Conference on Probabilistic Safety and Management, International Association for Probabilistic Safety Assessment and Management, pp. 1609-1614, 1998.

[15] M. S. Feather, S. L. Cornford, "Quantitative risk-based requirements reasoning," Requirements Engineering, 8(4), pp. 248-265, 2003.

[16] F. Bachmann, L. Bass, M. Klein, "Illuminating the fundamental contributors to software architecture quality," Technical Report CMU/SEI-2002-TR-025, Software Engineering Institute, Carnegie Mellon University, 2002.

[17] R. Kazman, L. Bass, G. Abowd, M. Wedd, "Saam: A method for analyzing the properties of software architectures," In Proceddings of The 16th International Conference on Software Engineering, Sorrento, Italy, 1994.

[18] M. Klein, R. Kazman, "Attribute-based architectural styles," Technical Report CMU/SEI-99-TR-022, Software Engineering Institute, Carnegie Mellon University, 1999.