

Retrospection and Perspectives on Pragmatic Software Architecture Design: An Industrial Report

Xiaofeng Cui

Abstract It is commonly recognized that the research on software architecture has enjoyed a golden age of innovation and concept formulation, and began to enter the mature stage of utilization. It is a natural expectation at present that the relevant concepts and methods have been populated and the improvements have been achieved in practice. In this paper, we give a retrospective report on our extensive architecture design of a large scale mission-critical system conducted in our company. The project has been trying to incorporate newly proposed concepts and methods from academic realm, and keeping introspective during the whole process. At the end of the project, we considered it as a successful architecting conduct, and believed that much experience can be gained from it. We describe the process, approaches, and results of this industrial practice. Experience and perspectives from the practitioners' points of view are analyzed and summarized. The contribution of this paper is twofold. Firstly, it presents a real-world software architecture design phenomenon, shares first-hand experience and perspectives on the pragmatic architecture design practices, therefore provides some insights into the state of the practice. Secondly, it may inspire further research on more effective and efficient methods and tools for better practices.

Keywords Software architecture design • Architectural significant requirements (ASR) • Architecture design decision (ADD) • Architecture design rationale

1 Introduction

Software architecture has emerged as a distinct discipline for about three decades [1], and enjoyed a golden age of innovation for more than 10 years [2]. Clements and Shaw [3] pointed out in 2006 that “*software architecture has been considered an unexceptional, essential part of software system building; the research in this*

X. Cui (✉)
Software Engineering Group, BACC, Beijing, China
e-mail: cuixf@pku.org.cn

field has enjoyed a golden age of innovation and concept formulation, and began to enter the more mature stage of quiet discipline and unremarkable utilization.”

In this situation, it is a natural expectation at present that the architecture-relevant concepts and methods have been populated, and the improvements have been achieved in practice. Much research however, indicates the gap between the state of the art and the state of the practice. As Moore et al. [4] pointed out earlier, “*solving a problem in theory and in practice are very different.*” In order to investigate the state of the practice of software architecture, there have emerged many empirical studies [5–7]. Most of them are based on surveys and drawing conclusions from interview feedbacks. The extensive first-hand reports from industrial practitioners are lack.

In this paper, we give a retrospective report on our extensive architecture design of a large scale mission-critical system conducted in our company for about 2 years. The project has been trying to incorporate newly proposed concepts and methods from academic realm, and keeping introspective during the whole process. At the end of the project, we considered it as a successful architecting conduct, and believed that much experience can be gained from it. Therefore we think it is worthy to summarize the experience on our interested problems about software architecture.

Based on our experience of industrial practice, we are interested in a series of questions to explore the pragmatic application of academic concepts and methods. Because of the space limitation, we concentrate on two of them in this paper:

- Q1: When, where, how and by whom are Architectural Significant Requirements (ASRs) identified? What are the main difficulties with the identification of ASRs in practice?
- Q2: How are Architecture Design Decisions (ADDs) made in practice? Are they made by intuition or formal procedures? What are the main difficulties with the architecture decision making in practice?

In this paper, we describe the process, approaches, and results of this industrial practice. Experience and perspectives from the practitioners’ points of view are analyzed and summarized. This report tries to present a real-world software architecture design phenomenon, share first-hand experience and perspectives on the pragmatic architecture design practices, and therefore provide some insights into the state of the practice. It also aims to inspire further research on more effective and efficient methods and tools for better practices.

The rest of this paper is organized as follows. Section 2 presents related work. Section 3 gives an overview of our architecting project. Section 4 describes in detail the requirements elicitation and architecture design deliberation of the project. Section 5 gives the retrospective analysis, including perspectives on the research problems and suggestions on better methods. Finally, Sect. 6 presents concluding remarks and future work.

2 Related Work

2.1 Architectural Significant Requirements

The inevitable intertwining of requirements and design has long been concerned by both academic and practice realms extensively [8]. At the architecture level, it is commonly conceived that some requirements, especially quality requirements (QR, or non-functional requirements, NFR), have crucial impacts on software architecture. These requirements are commonly called Architectural Significant Requirements (ASRs). Quite a lot of research has been conducted on the elicitation, identification, specification, and evaluation of ASRs [9].

Capilla et al. [9] highlight the importance and the role of quality requirements for architecting and building complex software systems. They point out that architecture challenges arise because the required quality attributes are informally stated during requirements elicitation and analysis. Niu et al. [10] argue that despite the increasing effort in engineering enterprise systems' requirements, little is known about the analysis of architecture interactions and tradeoffs. They propose a framework consisting of an integrated set of activities to help requirements analysis in practice.

While most research's focus is on NFRs, e.g., Niu et al. [10], Anish et al. [11] are interested in those Functional Requirements (FRs) that have significant impact on architecture, i.e. ASFR. They conduct a qualitative interview study aimed at identifying ASFRs' categories from various business domains. They also indicate that architects intuitively recognize ASRs and often seek out relevant stakeholders in order to ask Probing Questions (PQs) that help them acquire the information they need.

To systematically characterize ASRs, Chen et al. [12] present an evidence-based framework that consists of four sets of characteristics: Definition, Descriptive, Indicators, and Heuristics. They characterize ASRs as having wide impact on the system, exhibiting tradeoffs with other requirements, introducing constraints, breaking assumptions, and often being difficult to achieve.

Cleland-Huang [13] introduces a novel approach that uses "architecturally savvy personas" to depict quality concerns as personalized, architecturally significant user stories, and then leveraging these user stories to drive a system's initial design. The personas they created were intended to help reason about ASRs and to drive a series of architectural decisions.

2.2 Architecture Design Decisions

A series of methods have been proposed for the design or derivation of software architecture, e.g., Attribute-Driven Design (ADD) [14], Siemens Four-Views (S4V) [15], etc. Hofmeister et al. [16] compare five industrial software architecture design

methods and extract a general model, which classifies the kinds of design activities into architectural analysis, synthesis, and evaluation. Architectural analysis articulates ASRs. Architectural synthesis results in candidate solutions. Architectural evaluation ensures that the architectural decisions are the right ones.

The software architecture community has paid attention to design decisions and rationale for a long time. Hofmeister et al. [16] present a model of software architecture that consists of three components: elements, form, and rationale. Bosch [17] promotes that design decisions should be represented as first-class entities in software architectures. Tyree and Akerman [18] claim that a key to demystifying architecture products lies in the architecture decisions concept. Kruchten et al. [19] propose an ontology of software design decisions. Jansen et al. [20] develop a tool to model the software architecture as the composition of design decisions. Tang et al. [21] introduce the rationale-based model to support design rationale capture and traversal.

Falessi et al. [22] make a comparative survey on the decision-making techniques for software architecture design. They point out that the software engineering literature describes several techniques to choose among architectural alternatives, but it gives no clear guidance on which technique is more suitable than another, and in which circumstances. Their work represents a first attempt to reason on meta-decision-making, i.e., the issue of deciding how to decide.

Focusing on how architects make the transition from requirements to architecture, Heibisch et al. [23] argue that studies on the decision-making process examine the architecting process only after the architects have already established different architectural alternatives. They propose an approach for structuring the problem first in order to understand how appropriate solutions might look like, and creating architectural alternatives more directly from the requirements, enabling the evaluation of design decisions even before an architecture exists.

A lot of efforts have been made to cope with the interplay of requirements and architecture and their synergetic development [24, 25]. Chen [26] aims at advancing Requirements Engineering (RE) practice through the co-development of requirements and architecture by utilizing the relationship between ASRs and ADDs. They argue that in real software projects, requirements engineering is seldom separated from architectural design. However, the co-development of requirements and architecture is not well supported by current techniques and methods.

2.3 *Empirical Study on Architecture*

Kruchten and Stafford [1] pointed out in 2006 that “*The importance of software architecture for software development is widely recognized, yet transfer of innovative techniques and methods from research to practice is slow.*” In recent years, many empirical studies have been made on the state of the practice of software architecture.

Heesch and Avgeriou [5] give a report on findings of a survey that they have conducted with 53 industrial software architects to find out how they reason in real projects. The results are interpreted with respect to the industrial context and the architecture literature. They find that the greatest part of the participants does not seem to follow one particular architecture approach from the literature; instead they at least partially adopt architecture activities to define their own customized approach to architecture.

Anvaari et al. [6] conduct an exploratory study in Norwegian electricity industry, and point out that regarding the architectural decision-making process, most of the companies are not using well-known academic approaches, they are rather using their own procedures. They also show that the relationships among the actors of a software ecosystem could significantly affect the architectural-decision making process.

Dan et al. [7] point out that much research exists on architectural decisions, but little work describes architectural decisions in the real-world. They present the results of a survey with 43 architects from industry. They study characteristics of 86 real-world architectural decisions and indicate that dependencies between decisions and the effort required to analyze decisions are major factors that contribute to their difficulty. Good architectural decisions tend to include more decision alternatives than bad decisions. Finally, they found that 86% of architectural decisions are group decisions.

Daneva et al. [27] point out that only recently has RE research yielded the first five empirical studies of software architects' perceptions of QRs. To determine a possible range of views on how architects cope with QRs, they interviewed 20 software architects and found that software architects feel it's important to gain a deep understanding of the QRs and use this to deliver good architecture design.

Although there are such empirical studies, most of them are based on surveys and draw conclusions from interview feedbacks. This paper gives a detailed description and extensive analysis of a real-world architecting work conducted by ourselves, intending to provide some insights into the state of the practice, and inspire some points for further improvement.

3 Project Overview

3.1 *Project Background*

The project we study in this paper is an upgrading project of a large-scale aerospace flight control software system. The existing system has been used for about 8 years and carried out a series of missions successfully. The objective of the upgrading is to improve the system in terms of capability, usability, as well as new functionalities, so that it can serve well for the upcoming missions which will be much more complex and challenging.

In order to achieve the objective, our company set up an architecting team in 2013. The team comprised 11 experienced engineers. All of them are familiar with the existing system, and most of them have taken part in the development of the existing system. Because the system is quite large (6 subsystems, 21 configuration items, about 2000KLOC for the existing system; 11 subsystems, 40 configuration items, anticipated 3000KLOC for the new system), and the objective of this upgrading is ambitious, our company decided to grant the team 2 years (part-time job) to accomplish this architecting work.

By the end of 2015, the team finished its work and submitted the final architecture design of the new system. According to the review of the design and the feedback from the following development work in 2016, the whole process of architecting is considered to be successful and the design results are considered to be satisfactory.

3.2 Process and Methods

From the beginning of the architecting work, we have been trying to incorporate new concepts and methods promoted by the software architecture research community, and keep the architecting work effective, efficient, and pragmatic at the same time. The whole process of architecting is the synergetic deliberation of requirements and architecture solutions. For the sake of clarity, we roughly divide all the activities into 3 parts, as shown in Fig. 1 (which by no means implies a waterfall process of 3 stages). Table 1 summarizes the main approaches and points adopted in each part of the architecting, which will be explained in more detail in the next section. As mentioned in the introduction section, we concentrate on the “early part” of architecting in this paper, while omit other activities such as architecture documentation, evaluation, etc.

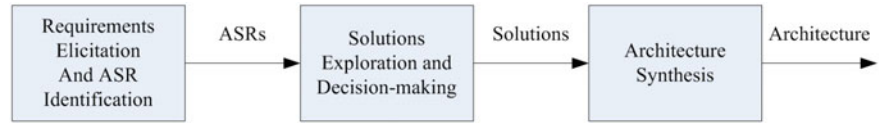


Fig. 1 Three parts of architecture design activities

Table 1 Approaches adopted in each part of the architecture design

Architecture design activities	Approaches and main points adopted
Requirements elicitation and ASR identification	Improvement points analysis of existing system; Requirements investigation of future missions
Solutions exploration and decision-making	Group deliberation; Group decision-making; Goal-oriented analysis
Architecture synthesis	Structure composition; Global design problems resolution; Key design decisions articulation

4 Architecting Process Description

4.1 Architectural Requirements Elicitation

According to literature and our experience, architects' thorough understanding of requirements is crucial to the success of architecting. Therefore in this project, we didn't split our team into requirement engineers and architecture designers. Our architecting team is at the same time requirement analysis team.

Based on the motivation of this upgrading project, we firstly defined 6 goals as the high-level goals of new system, which are 3 "high" (high performance, high reliability, high availability) and 3 "easy" (easy build, easy verify, easy use). Although various business functionalities (e.g., data processing, commanding and control, etc.) and other quality attributes (e.g., security) are important to the system as well, our analysis shows that the aforementioned 6 goals are crucial to the ultimate objective of new system. Table 2 lists and explains these goals in detail.

Then we made clear two main sources of the requirements for this architecting work: the improvement points of existing system, and the key demands of the future missions. In other words, we understood that the ultimate objective of our architecting work is to establish an architecture that can support the dramatic improvement of the existing system, as well as satisfy the new requirements of the foreseeable future missions.

In order to find out the improvement points of existing system, we conducted a thorough drawback analysis, a survey of the improvement appeals from its operators, and a deep statistical analysis of problem reports during its service history. In order to understand the demands of future missions, we made an investigation on their key characteristics. Although the information of future missions cannot be acquired comprehensively yet, the mission profiles can act as good inputs for the

Table 2 Design goals of the new system

Goals	Explanation
High performance	The future missions demand much higher performance requirements on the new system, including time performance (real-time) and space capability (throughput)
High reliability	The new system needs to have higher reliability than the existing system, so that it can satisfy the demand of critical missions
High availability	The new system needs to work correctly for longer time than the existing system, and support online maintenance
Easy to build	The new system needs to be built more easily than the existing system, which means requiring fewer human efforts and less time
Easy to verify	The new system needs to be verified easily, which means the verification cost can be decreased and quality can be guaranteed more easily than the existing system
Easy to use	The new system needs to have higher usability. The user experience can be increased and the operation efficiency can be increased dramatically

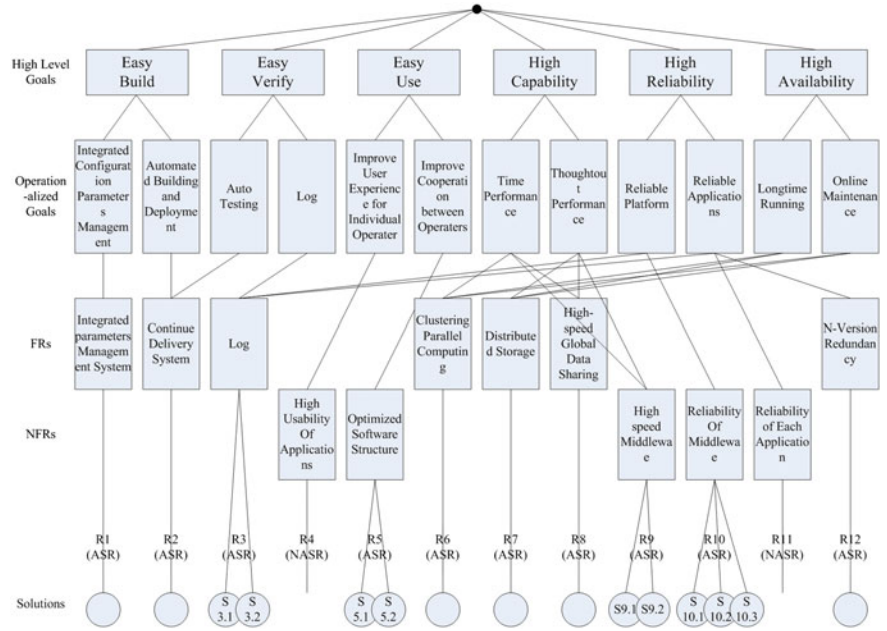


Fig. 2 The tree of goals, requirements, and solutions

architecture design, e.g., the highly volume data throughout, the long-lasting mission duration, etc.

When we had found out the improvement points and new requirements, we established functional requirements and non-functional requirements of the new system. All the requirements are linked to the high-level goals of the new system. The intermedium level contains operationalized goals, which bridge the requirements and high-level goals. The two-level goals and the requirements are depicted in Fig. 2 as a tree diagram (a simplified version for brevity). This diagram demonstrates the supporting relationships between the goals and requirements at different levels.

The last step of requirements analysis is to identify which requirements are architectural significant. We identified ASRs by the following criteria: if one requirement can be resolved within one configuration item of the system, or if they can be implemented at the detailed design or coding stages, then they are not architectural significant (i.e., NASR). On the contrary, if one requirement leads to a global design, or its design impacts more than one configuration items in the system, then it is an ASR. Figure 2 also depicts the results of our ASR identification, where 10 requirements are identified as ASRs, and 2 are identified as NASRs.

4.2 *Architecture Solutions Deliberation*

As the ASRs were identified, our next work was to resolve them. Although there are many literature-proposed methods for step-wise design solution exploration and formal decision making, we prefer to accomplish these jobs based on team expertise and group deliberation. Based on our experience in many projects, we think the latter approaches are more efficient and pragmatic.

We found two practical problems in this work and tackled them with distinct principles. Firstly, how can we resolve the requirements with their interrelationships in concern? Our principle is: if the ASRs are independent to each other, they can be resolved independently; if two or more ASRs are interdependent, then we combine them together and resolve them as one ASR. Secondly, how many solutions do we need to explore for one requirement? Our principle is: if we can find one solution and the design group can agree on it as an ideal solution, then we stop exploring more other solutions; if we cannot reach a consensus on the solutions of one requirement, which means we have not yet find the ideal solution, then we explore more solutions and make decisions on the explored solutions.

Considering the ASRs in Fig. 2, we found that the solutions to R1 and R2 are to newly build an Integrated Parameters Management subsystem and a Continue Delivery subsystem. We did not think it is necessary to explored more solutions for R1 and R2. The requirements R6, R7, R8, R12 are likewise. On the contrary, for the requirements R3, R5, R9, and R10, we found various design options and then explored more than one solutions for them (including the existing solutions). We explain these design and decisions as well as rationale in more detail as follows.

- Example 1: Requirement R3 and its solutions S3.1 and S3.2

The goal of R3 is to provide a log service with high capability, so that applications in the system can log a large amount of running information used for status monitoring, exception detecting, and root cause analysis. We explored two solutions for this requirement: S3.1 and S3.2, as shown in Fig. 3. Solution S3.1 is the existing solution of log service, where all log messages are sent to a remote monitor directly. This solution works well if the amount of log messages is not very large. But if we want the applications to report as much as information, the network traffic can increase dramatically and may block the running of applications, which is not tolerable for real-time applications. For this reason, we explored a new solution S3.2, where the log messages are sent to a local agent. The agent can buffer the log messages and send them to the monitoring and storage sites. This solution supports large amount of log messages because the agent can buffer the messages so that the applications need not worry about delay. In addition, the logs are stored in a dedicated site, so that they can be mined afterwards. Because S3.2 can satisfy the goal of R3 in a more effective way than S3.1, we can make a decision selecting S3.2.

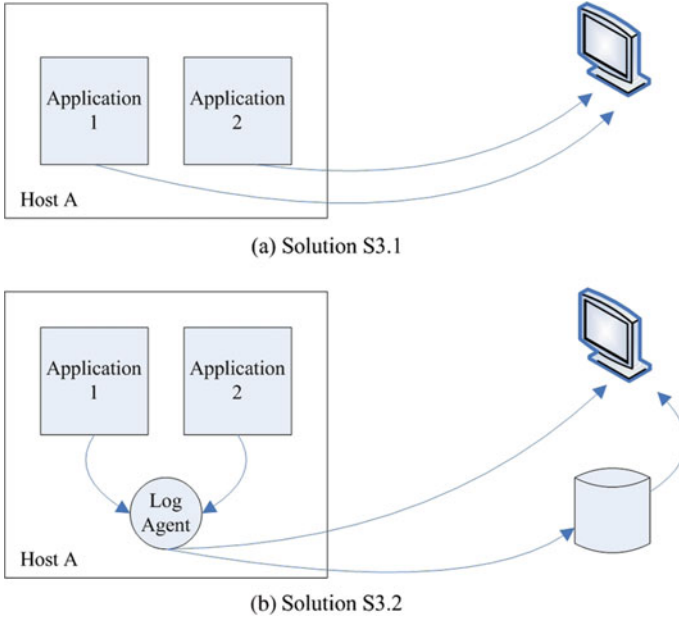


Fig. 3 Solutions to the requirement R3

- Example 2: Requirements R5 and its solutions S5.1 and S5.2

The goal of R5 is to improve the structure of existing system so that it has higher cohesion and looser coupling characteristics, and improve the efficiency of human operation. There are more than one improve points for this purpose. We pick up one of them as an example here. Figure 4 depicts two solutions for this purpose: S5.1 and S5.2. Solution S5.1 is the existing design of the planning and scheduling functionalities in the system, where the Planning software and Scheduling software belong to two different sub-systems. The Planning software outputs the plan files, which are sent to the Scheduling software as inputs. The Scheduling software schedules the applications in the system according to the plans. Based on this design, two operators are needed, one carrying out the planning operation and another carrying out the scheduling operation. Because the goal of R5 is to improve the operating efficiency, we explored a new solution S5.2, where the Planning software and Scheduling software are put into one new subsystem, i.e. the Planning and Scheduling subsystem. Then the interface between the two functionalities becomes an internal interface, which is easier to maintenance. In addition, only one operator is needed to operate the subsystem. Therefore we can make the design decision selecting the solution S5.2.

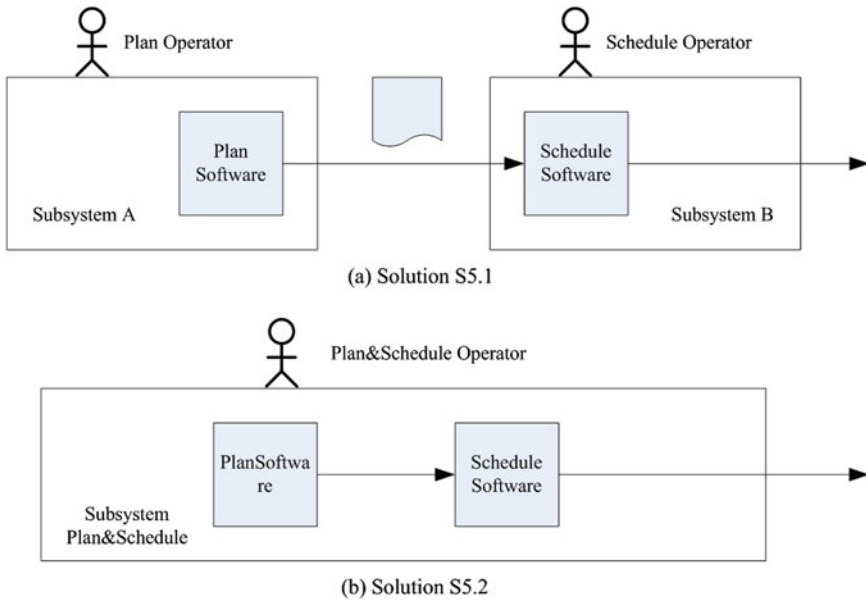


Fig. 4 Solutions to the requirement R5

- Example 3: Requirements R10 and its solutions S10.1, S10.2 and S10.3

The goal of R10 is to improve the reliability of the whole system. As the system needs to carry out multiple missions at the same time, we use the “Separation Policy” to improve reliability. There are different solutions to implement this requirement. We explored three of them: S10.1, S10.2, and S10.3, as shown in Fig. 5. S10.1 depicts one solution where one middleware instance supports multiple missions at the same time. This solution is straightforward, but has two drawbacks. The first one is that the middleware needs to suffer high volume of messages of multiple missions. The second one is that if one mission has an exception and make the middleware fail, e.g., blocked, then the other mission will be impacted. Therefore this solution is not a highly efficient and reliable solution. The solution S10.2 provide each mission one instance of the middleware, so that the fault in one mission cannot impact the other missions. Moreover, the burden of each middleware instance is decreased. This is a highly reliable solution. However, S10.2 cannot support inter-mission interactions which are necessary in some circumstances. Then we explored the solution S10.3, where multiple instances of middleware are established, and each application connects to one or more middleware instances based on their communication needs. This solution have the advantages of both S10.1 and S10.2, therefore it is the selected solution for R10.

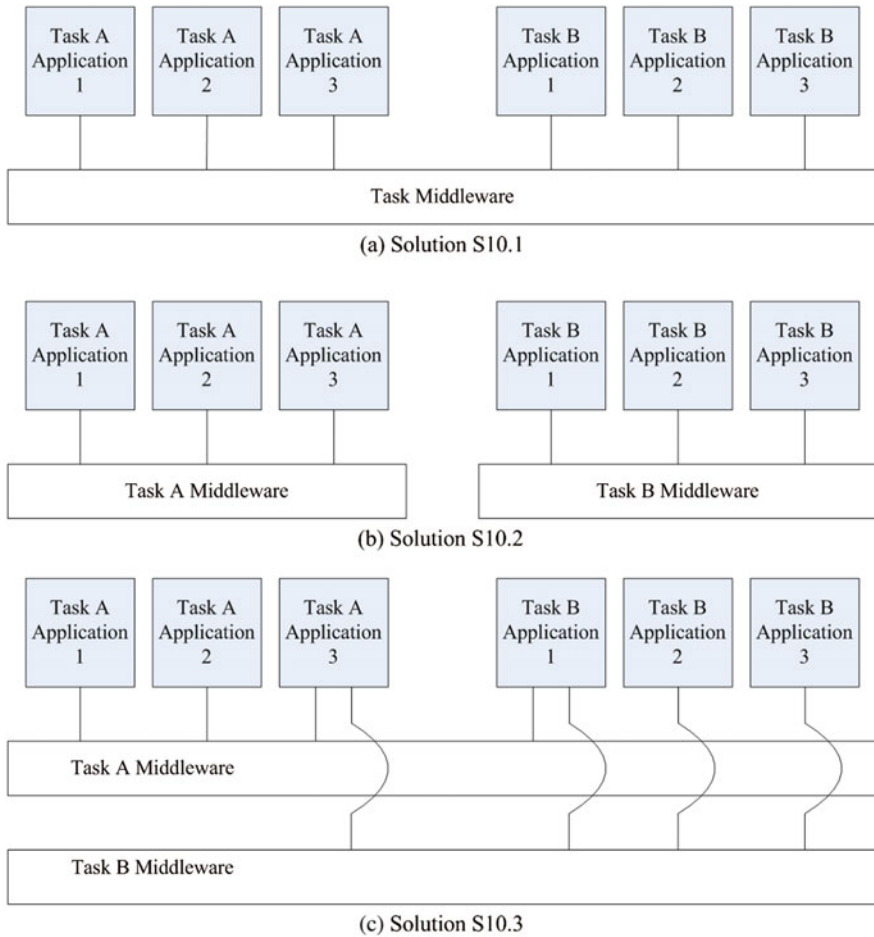


Fig. 5 Solutions to the requirement R10

4.3 Architecture Synthesis

The solutions for the ASRs are solution segments of the architecture. The final part of architecting work is to synthesize the whole architecture. In our opinion, the architecture of the system consists of: (1) the top one or two levels of the division of system (subsystem and configuration items), (2) the interfaces among these divisions, and (3) the global design decisions (ASRs and their solutions, decisions and rationale). For the sake of brevity, we omit the articulation of whole architecture design results here.

5 Retrospective Analysis

5.1 Perspectives on the Research Questions

Based on our experience on this project (as well as other projects we have carried out), we sum up our perspectives on the questions mentioned in Sect. 1. Table 3 lists the questions as well as both perspectives from literature and our practice.

For question Q1, our experience shows that architects in practice do not worry about the identification of ASRs indeed. We think a good architecture need to be designed upon the consideration of all FRs and NFRs, and only architects rather than requirements providers can figure out what requirements need to be considered at architecture level, while others can be solved at detailed design or coding level. Therefore ASRs are the outputs of architecting, rather than the inputs of it. It is commonly considered that the communication of ASRs from requirements field to architecture field is one major obstacle to architecture design. We think the

Table 3 Questions and perspectives

Questions	Perspectives from literature	Perspectives from our practice
Q1. When, where, how and by whom are Architectural Significant Requirements (ASRs) identified? What are the main difficulties with the identification of ASRs in practice?	ASRs are identified in the requirements field, and passed to architecture design field by stakeholders. The identification and communication are usually obstacles to architecture design <i>“Stakeholders and requirements engineers frequently fail to express or effectively communicate ASRs to the architects, preventing the architects from making informed design decisions.”</i> [12]	ASR identification is naturally an integral part of the architects’ job, because only the architects know what is architectural relevant. The pragmatic architects do not feel the difficulty of ASR identification. They instead think about all the perceived FRs and NFRs, and find out key issues need to resolve at architecture level
Q2. How are Architecture Design Decisions (ADDs) made in practice? Are they made by intuition or formal procedures? What are the main difficulties with the architecture decision making in practice?	ADDs are expected to be made by a systematic and formal methodology. The experience-based decisions are supposed to be irrational decisions <i>“Architects usually rely on their intuition and experience to create and choose between architecture alternatives. The viability of a decision can be checked by evaluating the architecture only after it has been created.”</i> [22]	For the sake of efficiency, the formal methods are not easy to be embraced in practice. But practical design activities can benefit from the decision-driven methods. These methods make architects explore the solution space thoroughly and make comparison explicitly, which can clearly improve the quality of design results

identification of ASRs is the responsibility of architects whereas not requirements engineers. In this situation, the communication of ASRs is not a real problem in practice.

For question Q2, our main point is that the consensus-based group deliberation is the effective and efficient design decision-making paradigm in practice. It is more pragmatic than formal or automated decision-making under most circumstance, because work efficiency and subjective consensus are critical factors of success in real-world projects. In our experience, a good design needs to satisfy (at least nearly) all the design participants. If one participant strongly resists one solution, there must be some problems with it. If one group cannot select a solution that satisfies all participants, they'd better explore more new solutions. On the contrary, when using formal decision-making methods, the result may be just the candidate with the highest score, which doesn't necessarily mean it can satisfy the group well.

5.2 What We Suggest for Better Methods and Tools

In our experience, there are still needs for effective architecting methods and tools in practice. In our opinion, many existing methods are not ideal because they have not concerned difficulties in real-world projects, e.g., the labor or time cost of the method, the importance of subjective approval, etc. We highlight two suggestions for better methods and tools in practice.

- Recording of the design process, decision, and rationale

During the process of our design and decision-making, we found that the real difficulty of large architecting projects is not the solution exploring or decision-making, because these jobs depend on the expertise of designers, and good designers should have ability to carry out these tasks well. The real difficulty, however, is the lack of efficient means to record the thoughts, arguments, rationale in the whole design processes. For large projects, the number of design and decision points can be several dozens and even more than one hundred (while only very few of them are mentioned as examples in this paper). During the deliberation process, it is not easy to record these kinds of information totally and precisely, because people think and talk faster than write. Even if we can record everything with electronic recorders, it is even not easy to find desired information from them because there is such long time duration and we cannot search it with keywords or other clue. In academic case studies or demo examples, all kinds of information can be recorded and described delicately. But for real-world projects, especially large ones, the schedule is tight and human resources are valuable, how to tackle the recording difficulty is a practical problem.

- Analysis of the architectural relevance of software defects

As aforementioned, in order to find out the drawbacks of the existing system, we considered the problem reports of existing system during its service history as

invaluable assets to inspire improvements. But we found it is not easy to analyze the architectural relevance of some problems. Is the problem implying an architecture drawback, or is it only a lower-level design or implementation defect? The problems need to be analyzed one by one. For large volume of reports, it is a high burden of human efforts. If there are systematic methods for the problem reporters (usually users, programmers, etc.) to identify one problem's architectural relevance correctly in the first place, the conduct of architecture upgrading will be much more effective and efficient.

6 Conclusions and Future Work

The software architecture realm has enjoyed its golden age and is expecting blooming and exciting application effects. We have conducted a series of software architecting work in our company, trying to incorporate new concepts, principles, and methods to improve the architecting results. We make a respective report on a large-scale and extensive architecting project, elaborating the process and methods involved, comparing perspectives from literature and practice, and presenting our suggestions on the future methods for more effective and pragmatic application.

The report is not a thorough empirical study, and the result only represents one kind of circumstance, but we think it can contribute some ideas for further research and practice. Because of the space limitation, this paper only concentrates on some aspects of architecting, i.e., ASRs and ADDs. We plan to embrace more methods in the future architecting practices, e.g., architecture documentation and evaluation, and make more thorough analysis on the practice experience.

References

1. Kruchten, P., Stafford, J.: The past, present, and future for software architecture. *IEEE Softw.* **23**, 22–30 (2006)
2. Clements, P., Shaw, M.: The golden age of software architecture revisited. *IEEE Softw.* **26**, 70–72 (2009)
3. Shaw, M., Clements, P.: The golden age of software architecture. *IEEE Softw.* **23**, 31–39 (2006)
4. Moore, M., Kazman, R., Klein, M., Asundi, J.: Quantifying the value of architecture design decisions: lessons from the field. *Int. Conf. Softw. Eng.* 557–562 (2003)
5. Heesch, U.V., Avgeriou, P.: Mature architecting—a survey about the reasoning process of professional architects. In: Ninth Working IEEE/IFIP Conference on Software Architecture, pp. 260–269 (2011)
6. Anvaari, M., Conradi, R., Jaccheri, L.: Architectural decision making in enterprises: preliminary findings from an exploratory study in Norwegian electricity industry. *Software Architecture*, pp. 162–175 (2013)
7. Dan, T., Galster, M., Avgeriou, P.: Difficulty of architectural decisions—a survey with professional architects. *Eur. Conf. Softw. Archit.* 192–199 (2013)

8. Swartout, W., Balzer, R.: On the inevitable intertwining of specification and implementation. *Commun. ACM* **25**, 438–440 (1982)
9. Capilla, R., Babar, M.A., Pastor, O.: Quality requirements engineering for systems and software architecting: methods, approaches, and tools. *Requirements Eng.* **17**, 255–258 (2012)
10. Niu, N., Xu, L.D., Cheng, J.R.C., Niu, Z.: Analysis of architecturally significant requirements for enterprise systems. *IEEE Syst. J.* **8**, 850–857 (2014)
11. Anish, P.R., Daneva, M., Clelandhuang, J., Wieringa, R.J., Ghaisas, S.: What you ask is what you get: understanding architecturally significant functional requirements. *Requirements Eng. Conf.* 86–95 (2015)
12. Chen, L., Alibabar, M., Nuseibeh, B.: Characterizing architecturally significant requirements. *IEEE Softw.* **30**, 38–45 (2013)
13. Cleland-Huang, J.: Meet Elaine: a persona-driven approach to exploring architecturally significant requirements. *IEEE Softw.* **30**, 18–21 (2013)
14. Bass, L., Clements, P., Kazman, R.: *Software Architecture in Practice*, 3rd edn. Pearson (2013)
15. Hofmeister, C., Nord, R., Soni, D.: *Applied Software Architecture*. Addison-Wesley (2000)
16. Hofmeister, C., Kruchten, P., Nord, R.L., Obbink, H., Ran, A., America, P.: A general model of software architecture design derived from five industrial approaches. *J. Syst. Softw.* **80**, 106–126 (2007)
17. Bosch, J.: Software architecture: the next step. In: *1st European Workshop on Software Architecture (EWSA'04)*, pp. 194–199 (2004)
18. Tyree, J., Akerman, A.: Architecture decisions: demystifying architecture. *IEEE Softw.* **22**, 19–27 (2005)
19. Kruchten, P.: An ontology of architectural design decisions in software-intensive systems. In: *2nd Groningen Workshop on Software Variability Management*, pp. 54–61 (2004)
20. Jansen, A., Jan, V.D.V., Aygeriou, P., Hammer D.K.: Tool support for architectural decisions. In: *IEEE/IFIP Conference on Software Architecture*, vol. 4 (2007)
21. Tang, A., Jin, Y., Han, J.: A rationale-based architecture model for design traceability and reasoning. *J. Syst. Softw.* **80**, 918–934 (2007)
22. Falessi, D., Cantone, G., Kazman, R., Kruchten, P.: Decision-making techniques for software architecture design: a comparative survey. *ACM Comput. Surv.* **43**, 88–87 (2011)
23. Hebisch, E., Book, M., Gruhn, V.: Scenario-based architecting with architecture trace diagrams. In: *IEEE/ACM 5th International Workshop on the Twin Peaks of Requirements and Architecture (TwinPeaks)*, pp. 16–19 (2015)
24. Cleland-Huang, J., Hanmer, R.S., Supakkul, S., Mirakhorli, M.: The twin peaks of requirements and architecture. *IEEE Softw.* **30**, 24–29 (2013)
25. Galster, M., Mirakhorli, M., Cleland-Huang, J., Franch, X., Burge, J.E., Roshandel, R.: Towards bridging the twin peaks of requirements and architecture. *ACM Sigsoft Softw. Eng. Notes* **39**, 30–31 (2014)
26. Chen, F.: From architecture to requirements: relating requirements and architecture for better requirements engineering. *Requirements Eng. Conf.* 451–455 (2014)
27. Daneva, M., Herrmann, A., Buglione, L.: Coping with quality requirements in large, contract-based projects. *IEEE Softw.* **32**, 84–91 (2015)



<http://www.springer.com/978-3-319-60169-4>

Computer and Information Science

Lee, R. (Ed.)

2018, XIV, 234 p. 96 illus., 50 illus. in color., Hardcover

ISBN: 978-3-319-60169-4