Software Architects in Practice: Handling Requirements

Vidya Lakshminarayanan, WenQian Liu*, Charles L Chen, Steve Easterbrook*, Dewayne E Perry

Empirical Software Engineering Lab
Electrical and Computer Engineering
The University of Texas at Austin, TX
{vidya,clchen,perry}@ece.utexas.edu

*Software Engineering
Department of Computer Science
University of Toronto, Canada
{wl,sme}@cs.toronto.edu

Abstract

Software architecture can be a critical factor in software development. Understanding what software architects do in practice is necessary to the enterprise of providing techniques, methods, process, tools and technologies to support the development and use of software architecture. In this paper, we present the results of how architects handle requirements in practice. We then summarize the key lessons learned from the study.

1 Introduction

Research in the areas of requirements engineering and software architecture have developed independently for more than a decade [4][7]. However, in the last few years, a number of researchers have begun to investigate the relationships between requirements and architecture, and how to bridge the gap from one to the other [1][3]. In particular, techniques have been proposed for generating software architectures from the requirements [2][5][9][10]. However, there have been no studies to date on how architects currently perform this transformation in practice.

In this paper, we report the some of the results of a multiple case study of experienced software architects. We take an empirically based approach and use an interview-based case study methodology to carry out our investigations. Case studies are a specific empirical research method used to gain a deep understanding of a particular phenomenon in its real life context. As such, they are characterized by analytical generalizations rather than statistical generalizations; they are to be understood in terms of analysis and comparison of cases, not of samples [11].

So far, we have interviewed fourteen different architects from different domains (such as security, usability, product lines, etc.) in different organizations (ranging from relatively small to extremely large, from specific product focuses to general system and solution providers, etc).

The architects in this study were chosen from software intensive organizations in two general areas: in and near Toronto, and Texas (Austin, Houston and Dallas). Where the subject architects are from the same companies, they either represent different divisions (which are often the size of medium sized companies) or represent different kinds of architects. With a few exceptions, our subject architects have significant experience both in their roles as architects (4 to 15 years). Where the subjects have only a few years experience as architects, they do have significant experience as developers (five or more years).

In [6], we describe the full scope of our study, and our process of defining it from its preparation to its evidence chain and evidence trail. We also discussed various validity issues with the then current state of our case studies. Our main concern then was the concentration of architects from one international company. As noted above, we have expanded the representation of architects in terms both of companies and of domains. Thus, we feel that the external validity of our multiple case study has been strengthened significantly.

Copyright © 2006. Lakshminarayanan, Liu, Chen, Easterbrook and Perry. Permission to copy is hereby granted provided the original copyright notice is reproduced in copies made.

2 Requirements

From our data, it is evident that there was a clear consensus on what requirements are - things that are wanted, needed, asked for, or demanded. However, there was little consensus on how they are handled.

2.1 Functional vs. Non-Functional

A typical distinction is that functional requirements specify the functions that a system must have, while non-functional requirements describe general constraints on the acceptable solutions, such as performance requirements, software design constraints, software quality attributes, etc.

Our subjects do not have a uniform opinion about functional and non-functional requirements. The majority of our subjects distinguish between them. A few architects, however, suggested that they see no real difference between functional and non-functional requirements and each requirement (if committed) must be accommodated in the design. For example, one subject indicated that security is neither functional nor non-functional; it is simply a serious business and technical requirement that can "*make or break the deal*".

Several subjects consider that every requirement has both functional and non-functional aspects. Depending on one's perspective and the problem context, one aspect may dominate the other. In security, there are standard functional requirements dealing with interoperability and manageability issues: a VPN client has to implement a chosen protocol in order to communicate with the server; a system administrator needs to be able to safely update the access information using scripts overnight. Also, specific requirements are set to prevent performance degradation: one cannot impose more than $^{1}/_{10}$ of a second latency on the startup of a connection; or more than 3% throughput overhead due to cryptography.

While there is some disagreement, the differences between them are more cosmetic than fundamental. The underlying themes of our subjects' opinions are as follows:

- Non-functional requirements are frequently used for convenience to refer to high-level properties and abstract phenomena that are desirable.
- Non-functional properties must be (re)formulated as functional ones before they can be satisfied.

2.2 Discovering *Real* Requirements

Requirements set the goals and expectations for a system to be developed. However, a variety of requirements problems make it difficult to produce the right system. Our subjects state that requirements elicitation is difficult because multiple stakeholders are usually involved. Each one has a view of what is important and what needs to bee done based on past experiences, and personal prejudices. Requirements are often expressed in technological terms and often reflect their personal biases and, thus, obscure the *real* requirements.. Higher level goals are often left out, leading to partial or incorrect requirements. Our subjects recognize this as a problem and respond to it by participating in the elicitation process. This helps them obtain first hand the information to understand what the stakeholders actually need.

Sometimes architects with their experiences and knowledge need to influence and educate the stakeholders to broaden their focus and move in a different direction. One of our subjects, however, suggests that architects should communicate with the requirements engineers instead of going straight to the customers because of the following reasons: 1) these engineers have the necessary training for handling requirements and customers, and 2) the architects' understanding may differ from that of the requirements engineers leading to synchronization problems between the requirements and the architecture.

Another issue is that requirements are not always consistent. Interacting with the stakeholders (and possibly educating them) can help to resolve these inconsistencies. Our subjects often rely on brainstorming in a group setting to resolve such issues. This technique focuses on generating as many ideas as possible. The resulting ideas are evaluated, and the ones best suited for the stakeholders are chosen. Unfortunately, not all inconsistencies can be resolved this way. Occasionally, architects must rely on their own judgment to resolve these inconsistencies. This tactic is risky and not recommended but sometimes it is the only alternative available to them.

2.3 Anticipating Changes

Changes are a very common in software developments. They can represent new requirements or re-evaluations of existing requirements. They are usually stated as additional functionality or an

incremental problem to solve. One subject indicates that in creating an architecture that can be reused and can be adapted to changing needs, architects must be able to see the big picture so that resources can be better planned and critical problems can be better addressed. He further suggests that "*what is going to change really is related to the domain*" and trying to handle everything without an understanding where changes are likely to occur is a poor approach in software engineering. One key to anticipating changes is the understanding of business needs and customers goals. Otherwise, it will be just a guessing game.

Another subject suggests that satisfying every request from the customer may not be the right thing to do because some requests do not fit in the long-term direction of the industry. He believes that the key to understanding the long-term direction of industry must involve the collaboration of the industrial and research communities. By sharing their knowledge and experience, both communities can benefit.

Professional experience is always an asset.. Typically, experienced architects will seek to reuse information and predict possible problems and changes. Experience helps architects to identify potential risks, what is hard, and what are critical issues.

A number of subjects suggest that using good design principles provides an indirect solution for dealing with changes and anticipating future requirements. For example: 1) the natural cohesion found in the requirements can be used in the design to make maintenance and enhancements easier, 2) modularity can be used to minimize the effects of changes on the entire system, and 3) generic solutions can be used to provide reusable and extensible frameworks.

The benefits of anticipating changes do not come without a price or without challenges. First, it is a time consuming task. There may be cases when this investment is not warranted. This is especially true for products with shorter anticipated life spans and in cases where the time to market for the first release is a dominant factor.

Second, "disruptive technologies" (technologies that become dominant and radically change the state of practice) provide a special challenge. One way of dealing with this is to design the system to be adaptable to mitigate the risk of unforeseeable changes.

Third, it is always a challenge to be able to correctly interpret requirements in the absence of contextual information. In such cases, making incremental changes and integrating back into the product is appropriate. It also helps in uncovering any hidden assumptions.

Fourth, with a 'middle man' (such as marketing staff) filtering and interpreting the requirements, the real problem may not be conveyed properly to the architects. To mitigate this problem, one subject believes that architects need to analyze *"a number of problems in aggregate*" instead of focusing on a given single instance before committing to a solution.

One of the ways to manage change is to lay out a roadmap to help customers move on from one release of the system to the next. By having a clear plan for upgrades and changes, the problem of dealing with multiple versions with different configurations can be minimized or even eliminated.

2.4 Managing Requirements

Software architects use a number of strategies to manage complex and costly requirements. Understanding the rationale and the business problem of the given requirements by asking the 'why' and 'what' questions is critical. Answers to these questions provide both deeper insight into the real problem and more options for solving them. During these back and forth question-and-answer sessions, the negotiation of the requirements is actually taking place to produce acceptable requirements that can be signed off by both sides.

Conflicts, unrealistic 'sales promises' and impossible requirements must all be negotiated with the customer. In some cases, a compromise can be achieved, but in others, 'no' is the only answer. Under such circumstances, prioritization helps to determine which requirements are essential and which are optional. These priorities can be based on cost/benefit analyses or specific areas of concern to the stakeholders. There may still be a case that there are too many requirements all with the highest priority where an "*80/20 rule*" could be applied: i.e., choose 20% of the requirements that cover 80% of the stakeholders' concerns.

Problem decomposition is often used to manage complexity. In most problems that involve automating human activity, decomposition is fairly straightforward because people tend to do

things in uncomplicated ways.. Decomposition helps to break down the problem into smaller pieces and to make implementation easier. We note, however, that the refinement and decomposition methods and techniques used often depend on the architect's experience and domain knowledge.

Since there are no specific techniques or methods to handle complex requirements, it is important that the architects perform post mortem analysis so that they can analyze their mistakes and come up with better methods and techniques to handle such requirements. However, such post mortem analyses are usually not conducted well in practice because of a lack of tool support to trace back to the requirements.

3 Lessons Learned

Lessons learned from out subjects include:

Lesson 1 *Architects need to be generalists rather than specialists. They need to draw from a wide range of design and domain knowledge.*

Lesson 2 *The distinction between functional and non-functional requirements is not always a concern, but both must be carefully considered and reflected in the architecture.*

Lesson 3 *Change is inevitable in software developments. Therefore, anticipating changes early can save implementation and maintenance effor Heuristics and professional experience play a large role in this.*

Lesson 4 *Requirements are often filtered by middle men where much of the relevant contextual information is missing. This can cause major problems for the architect.*

Lesson 5 *Architects need to ensure that they are working with reasonable, consistent requirements. If there are problems with the requirements, then they should either send the requirements back to the requirements engineers for rework, or they should negotiate directly with customers. Most of our subjects favored the latter approach.*

4 Future Work

There are still lessons to be obtained from our architects beyond what we have provided here. We expect these to be made available in workshop, conference and journal publications. We will also make available appropriately sanitized technical reports summarizing the results of our interviews and providing salient quotes [8].

Acknowledgements

We thank our anonymous interviewees and their companies for their contributions. This research is supported in part by NSF CISE Grant CCR-0306613 and IBM CAS Fellowship. We also thank our co-researcher Deepika Mahajan

References

[1] D. M. Berry, R. Kazman, and R. Wieringa, editors, *The Second International Software Requirements to Architecture Workshop (STRAW'03)*, ICSE 2003, Portland OR.

[2] M. Brandozzi and D. E. Perry, "From Goal-Oriented Requirements to Architectural Prescriptions: The Preskiptor Process", in [3].

[3] J. Castro and J. Kramer, editors, *The First International Workshop on From Software Requirements to Architecture (STRAW'01,)*.ICSE 2001. Toronto.

[4] D. Garlan, "Software architecture: a roadmap", *Future of Software Engineering*, ICSE 2000, Limerick Ireland,.

[5] D. Jani, D. Vanderveken and D. E. Perry. "Deriving Architectural Specifications from KAOS Specifications: A Research Case Study", *European Workshop on Software Architecture* 2005, Pisa Italy.

[6] W. Liu, C. L. Chen, V.Lakshminarayanan, D.E. Perry, "A Design for Evidence-based Software Architecture Research", *Workshop on REBSE'2005*, ICSE 2005, St. Louis.

[7] B. Nuseibeh and S. Easterbrook, "Requirements engineering: a roadmap", *Future of Software Engineering*, ICSE 2000, Limerick Ireland.

[8] D. E. Perry. NSF Grant CCR-0306613 Project Web Site "Transforming Requirement Specifications into Architectural Prescriptions", http://www.ece.utexas.edu/~perry/work/projects/nsf-r2a/ [29 October 2005]

[9] A. van Lamsweerde, "From System Goals to Software Architecture", In M. Bernardo and P. Inverardi, editors, *Formal Methods for Software Architectures*, 2003.

[10] D. Vanderveken, A. van Lamsweerde, D. E. Perry, and C. Ponsard, "Deriving Architectural Descriptions from Goal-Oriented Requirements Models", September 2005

[11] R. K. Yin, *"Case Study Research: Design and Methods"*, 3/e. Thousand Oaks, CA: Sage Publications, 2002.