Acknowledgements

This has been a long journey. Thanks to the great and inspiring people that accompanied me on the adventure, I have enjoyed virtually every moment of it. I am grateful to many for making this doctoral trip such an enjoyable experience.

First of all, I would like to thank my advisors, Peter de With and Hans van Vliet. I started out under the guidance of Peter, who taught me how to write down my thoughts and experiences with the rigor and depth of academic work. After running into Hans at Philadelphia airport on our way to yet another WICSA conference, it quickly became clear that the focus of my work at Logica was very much aligned with Hans' work, resulting in a very productive three-way cooperation. It has truly been a pleasure working with you both, and I feel privileged that you have spent so much of your valuable and busy time answering my many questions and guiding me to this milepost.

One of my original objectives in writing a Ph.D. thesis was the hope that I could one day share my views with the authors in the field that I admired: people like Philippe Kruchten, Eoin Woods, Len Bass, Paul Clements, Rod Nord, Judith Stafford and Rick Kazman. To my delight, this objective was achieved long before the thesis was even half finished. I am thankful for the inspiration and friendship of these authors, along with other major contributors like Patricia Lago, Rich Hilliard, Christine Hofmeister, Antony Tang and the other members of IFIP Working Group 2.10. Thanks particularly to Paul for inviting me and Jos to his home, and to Christine for a great evening of Mozart string quartet playing.

Another pleasure was to cooperate in GRIFFIN and QuadREAD, two Jacquard projects that succeeded in bringing industry and academia closer together. I would particularly like to thank Paris Avgeriou, Klaas van den Berg, Roel Wieringa, Mehmet Aksit, Maia Daneva, Laura Ponisio, Viktor Clerc, Rik Farenhorst and Remco de Boer for the many hours of lively and productive discussions.

This thesis would not have been possible without the support of Logica. I am particularly grateful for the encouragement and support of my direct managers over the years: Hans Keizer, Peter Koger, Rob van der Stap, Robin Rijkers, Reg Hayes, Peter Hartog and André van der Wiel. I would also like to acknowledge the contributions of many other (ex-)colleagues who have inspired and challenged me: specifically Andrew Key, Bert Kersten, Laurens Lapré, Michiel Perdeck, Wouter Geurts, Herman Postema and Wouter Paul Trienekens. I would also like to thank Ben Voogel, whose remarks when I left after seven years of working for his MUIS Software company sparked my interest in academia again for the first time since graduating.

Finally, I would like to thank my family - especially my children Florine and Tycho and my wife Jos. You are a continuous source of inspiration to me.

Malden, April 2012

1 Introduction

As the presence of Information Technology increases, so grows the impact of the design decisions shaping the IT solutions that touch our lives. We feel this impact as we are amazed at the new possibilities offered by developments like the Internet, which all but redefined our social interaction experience within the time span of one generation. But the impact of IT design decisions is not always positive. We sometimes feel negative impact as small irritations, like our kids complaining whenever their favorite social media site modifies its functionality. Sometimes, however, things go really wrong, with far-reaching consequences.

Once in a while, a single wrong design decision makes its impact felt across an entire nation's political landscape, or even globally. In the past decade, the Netherlands alone has seen a number of such events:

- C2000, a newly designed communication system for emergency workers that did not allow proper communication from within buildings, and that failed in large-scale disasters [Expertgroep C2000, 2009].
- OV-chip, a brand new public transportation payment chip card whose encryption was so weak that it was breached as soon as it was publicly available, allowing people to modify the travel balance on their cards without paying [de Winter, 2011].
- Dutch highway tunnel safety systems, with software quality issues so severe that the tunnels were fully opened to the public years after the original deadline [Gram and Keulen, 2010].

What do these examples have in common? First of all, they are all highly visible projects in the public government sector, plagued by multiple issues. Second, in all three cases, is was not the functionality of the solutions that was wrong - it was the other, “non-functional” aspects: in these cases, capacity, security and quality. Third, in all three cases, there was a client/supplier situation, where the requirements specification (the “what”) was drawn up by the client and the solution design (the “how”) was created by one or more suppliers.

A review [Dalcher and Genus, 2003] estimated the financial cost of failed IT projects in the United States at US$150 billion per annum, with a further US$140 billion in the European Union. More importantly, the above examples show a significant impact on our quality of life. Some are even life threatening. If we want to address these problems, the IT industry and its clients need a better understanding of how to address non-functional needs. We need to better understand how to architect IT solutions that adequately serve their purpose, especially in client/supplier situations.

1.1 Context

This thesis is the result of a journey to improve architecting practices in Logica, a large IT Services company. This journey started in 2003, when we identified a need to better understand the impact of Non-Functional Requirements on our solutions and their delivery. It gained momentum and focus in 2006, when the company’s Technical Board expressed the requirement for a standard approach towards architecting across the company. This requirement gave us a clear sense of direction, and the result was the establishment and implementation of a solution architecting approach: Risk- and Cost Driven Architecture (RCDA).

Logica is an IT corporation of approximately 40,000 people across 40 countries. The company has a diverse business portfolio, consisting of services centered on business consulting, systems development and integration, and IT and business process outsourcing. Although the scope of the work presented in this thesis is the whole Logica group, the surveys described were limited to the Netherlands, which has 4500 employees. Within the company, a function called “technical assurance” is responsible for assuring the feasibility, suitability and acceptability of the solutions we offer our clients. The majority of the work in this thesis was done in the context of technical assurance in Logica Netherlands. The main activity of the technical assurance function is to review large and complex bids and delivery projects. The extensive interaction we had over the years with hundreds of IT projects with various degrees of size and complexity, in multiple industry sectors, is one of the main data sources for the research presented here. The lessons learned and insights harvested from these projects are scattered throughout the chapters of this thesis.

The global head of technical assurance also leads the group-wide Architecture

Community of Practice (ACoP), an informal international community of practicing architects. The ACoP is the second important source of data that contributed to this research, especially the surveys.

A third activity of the technical assurance function is to initiate improvements within the company, based on the lessons learned in the bids and projects we review. It is in this capacity that the drive towards a common architecture approach was initiated, culminating in the development and implementation of RCDA.

1.2 Key Concepts

1.2.1 Solution Architecture

The success of an IT Services company depends on its ability to make the right choices about the structure and behavior of the solutions it delivers to its customers. In the last decades, the IT industry has started calling this structure and behavior "architecture", in analogy to the building design domain. In the field of software engineering, the notion of "software architecture" is one of the key technical advances over the last decades [Farenhorst and de Boer, 2009]. In that period, there have been two distinct fundamental views as to the nature of architecture:

1. Architecture as a higher level abstraction for software systems, expressed in components and connectors [Shaw, 1990, Perry and Wolf, 1992].

2. Architecture as a set of design decisions, including their rationale [Kruchten, 1998, Jansen and Bosch, 2005, Tyree and Akerman, 2005].

View 1 is about "the system-level design of software, in which the important decisions are concerned with the kinds of modules and subsystems to use and the way these modules and subsystems are organized" [Shaw, 1990]. View 1 is focused on the choice and organization of components and connectors.

View 2, architecture as a set of design decisions [Jansen and Bosch, 2005], is more generic and has been beneficial to both the architecture research community and its practitioners [Tyree and Akerman, 2005]. This view of architect*ure* implies a view of architect*ing* as a decision making process, and likewise a view of the architect as a decision maker.

In Chapter 8, we will discuss our own view about the nature of architecture, which builds on and extends these two views of software architecture. Most the work presented here is based on practices and research that have emerged from the software architecture community of researchers and practitioners. The term software architecture, however, no longer covers the main subject of the work presented in this thesis. As an example, of the architects we have trained in the approach presented in Chapter 9, about half did not call themselves software architects. Their area of interest was often wider than software-intensive systems, covering business processes, IT infrastructure, information architecture, etc. They were interested in the approach because they had to architect solutions, and their architecting work involved all the same key concepts: stakeholders, concerns, decision making, etc. All of the “non-software”-architects indicated that the material presented was applicable to their area of interest.

The name we use for this spectrum of architecting disciplines is Solution Architecture, to indicate that the common denominator of these architecture disciplines is to find a *solution* to a particular set of stakeholders’ needs. This term is also used in the Enterprise Architecture domain: The Open Group Architecture Framework (TOGAF) [The Open Group, 2009] defines a Solution Architecture as *a description of a discrete and focused business operation or activity and how IS/IT supports that operation. A Solution Architecture typically applies to a single project or project release, assisting in the translation of requirements into a solution vision, high-level business and/or IT system specifications, and a portfolio of implementation tasks.* Although this definition is a little more specific than our notion encompassing various architecture genres, the focus on a single project and solution vision corresponds to our application of the term.

Our definition of Solution Architecture is based on the ISO 42010 definition of architecture: the *fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution.* [ISO 42010, 2011] This is a very comprehensive definition, and its scope depends very much on the interpretation of the word “system”. If by System is meant e.g. a complete enterprise, this definition pertains to an Enterprise Architecture. Solution Architecture is simply defined as the architecture of a solution addressing a particular set of stakeholder needs. This solution is the “system” in the ISO 42010 definition. The term Solution Architecture encompasses high-level solution shaping for most of the solutions built and operated by IT services companies: applications, service solutions, embedded systems, infrastructure, SOA implementation, systems integration etc. It can span infrastructure, information architecture, business processes and their environment. Thus our definition of Solution Architecture is narrower than the generic ISO 42010 architecture definition not so much in the scope of the system, but in the *raison d’être* of the architecture: to address a particular set of stakeholder needs.

1.2.2 Non-Functional Requirements

Requirements that describe *what* a solution should do are generally called functional requirements (FRs). There are, however, “other” requirements that are more closely related to *how* the solution should do what it does: how well, how fast, how reliably, etc. To distinguish these other requirements from the functional requirements, they are often called non-functional requirements or NFRs. This is admittedly not a very good name: rather than describe what its subject is, it says what it’s *not*. In the industrial context of the research presented here, however, the term NFRs is well established, and hence we have chosen to maintain it. Many terms closely related to NFRs are regularly used, such as “quality requirements” or “qualities” [Boehm and In, 1996, Bass et al., 2003], “attribute requirements” [Gilb, 1988], “performance attributes” [Gilb, 2005], “extra-functional requirements”, “system quality requirements” or even “-ilities”. [Mairiza et al., 2010] gives a nice overview and classification of the use of this terminology. In §2.3.2, we present our own classification of requirements used throughout this thesis. The classification distinguishes between two kinds of non-functional requirements: $NonFunctionalRequirements = QualityAttributeRequirements + DeliveryRequirements$, where *Quality Attribute Requirements* denotes the quantifiable requirements about solution quality attributes, and *Delivery Requirements* denotes the requirements on the delivery of the solution (such as when or by which means it should be delivered). The reader will notice that in some chapters, delivery requirements play a secondary role; specifically in Chapters 3 and 4 the term non-functional requirement is almost synonymous to quality attribute requirement.

NFRs represent a promising area for improvement, because dealing with NFRs is viewed as a particularly difficult part of requirements engineering [Berntsson Svensson, 2009]. Not properly taking NFRs into account is considered to be among the most expensive and difficult of errors to correct once an information system is completed [Mylopoulos et al., 1992] and it is rated as one of the ten biggest risks in requirements engineering [Lawrence et al., 2001]. NFRs are widely seen as the driving force for shaping IT systems’ architectures [Mylopoulos, 2006, Chung et al., 1999, Paech et al., 2002, Bass et al., 2003]. According to [Glinz, 2007], “there is a unanimous consensus that non-functional requirements are important and can be critical for the success of a project”.

In this thesis, we will be looking at how architects handle NFRs, and at how we can improve this handling in terms of solution design and communication between clients and suppliers of IT solutions.

1.3 Objectives

The research this thesis is based on was performed in a business context, and in the end the research objective always was a business goal: to improve the success of the company. This extremely high-level business goal is approached from several directions:

- The success of an IT Services company depends on its ability to make the right choices about the structure and behavior of the solutions it delivers to its customers.
- In Chapters 4 and 7, it is represented by the concept of IT project success. This concept is explored in detail in §4.2.1 on page 61.
- In Chapter 6, it is decomposed into the key aspects of consistency in delivery, risk management, customer satisfaction and knowledge incorporation for the purpose of defining an architecting process.

At the beginning of our journey, we suspected that an improvement in handling NFRs would contribute significantly to the company's business goals. More specifically, we encountered two practical challenges that needed addressing: how to structure a solution to address conflicting NFRs, and how to optimally quantify NFRs in customer-supplier relationships where stakeholder communication is limited for contractual or legal reasons. We also developed an interest in how architects perceive and handle NFRs, and if there is a relationship with project success. Hence, the first part of this thesis addresses the following research question and sub-questions:

RQ-1 How can Non-Functional Requirements be handled to improve the success of IT solutions and the projects delivering them?

(a) How can a solution be structured to best address conflicting Non-Functional Requirements?

(b) What is the best way to quantify Non-Functional Requirements across a contractual divide between customer and supplier?

(c) How do architects perceive and deal with non-functional requirements?

In 2006, the Technical Board expressed the requirement for a standard approach towards architecting solutions across the company. This standard approach would be subject to the requirements set by the CMMI®[1], an approach [CMMI Product Team,

[1]CMMI: Capability Maturity Model Integration®, registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

2010] for improving and assessing organizations' performance, containing ample material relevant to IT solution development. We started out to define a formal architecting process, but in 2009 decided that a framework of practices harvested from industry and literature would better fit the company's needs for a highly generic, customizable approach. The second part of this thesis documents this part of the journey, and addresses the following research question and sub-questions:

RQ-2 What is a good solution architecting approach to improve an IT service provider's success?

(a) What is the nature of solution architecting in the business context of a large IT services company?

(b) What requirements does an architecture process need to fulfill in order to comply with CMMI maturity level 3?

(c) How do architectural knowledge sharing practices relate to challenges in solution delivery projects?

(d) What architecting practices improve an IT service provider's success, and what guidance should they contain?

(e) What is the effect of training architects in such architecting practices?

1.4 Approach

In addressing the objectives identified above, we took a pragmatic approach. The work had to be done in the context of a busy and dynamic IT company, within the usual business constraints and pressures of such a real-life environment. This environment carried the benefit of giving us access to key resources that were used extensively in this research: the Architecture Community of Practice (ACoP) and the interactions with hundreds of active IT delivery projects in the context of technical assurance. On the downside however, due to the environment's constraints and pressures, the work is not the result of a carefully planned research program. Rather, it presents nuggets of related research that helped a business identify and implement some important improvements.

The thesis contains the results of three surveys among the architecture community, and one major case study. The approaches we present are partly based on insights harvested from projects that we interacted with; some of the projects are presented as small case studies, examples and anecdotal evidence. Other than the surveys and harvested insights, important contributors to the approaches are literature and analysis.

The research presented here is best classified as action research, since the primary researcher was an active participant in the studies, and the objective was to improve the subject of the studies and to generate knowledge at the same time [Kock, 2011]. According to [Davison et al., 2004], the five stages of Canonical Action Research are diagnosis (identifying a problem), action planning (considering alternative courses of action), action taking (selecting a course of action), evaluating (studying the consequences) and specifying learning (identifying general findings). These stages are clearly present in the work presented here: both surveys and the case study are diagnosis, and the approaches presented contain action planning and taking, combined with evaluating and identifying learning.

1.5 Outline

The structure of this work reflects the journey described above. The work is in two parts: part I is about dealing with non-functional requirements (NFRs), and part II is about establishing a solution architecting approach. Part I, Dealing with Non-Functional Requirements, starts with two chapters that each analyze an existing problem in dealing with NFRs in practice (RQ-1a and RQ-1b), and then presents a method for dealing with that problem. The third and final chapter in part I presents the result of a survey held to gain understanding of how architects deal with NFRs (RQ-1c). The chapters in part I are:

Chapter 2: *Resolving Requirement Conflicts through Non-Functional Decomposition.* We start out with a chapter on how NFRs (and especially conflicting NFRs) directly impact a solution's preferred structure. We build a framework that both provides a model and a repeatable method to transform conflicting requirements into a system decomposition. The chapter presents the framework, and discusses two cases onto which the method is applied.

Chapter 3: *Dealing with Non-Functional Requirements across the Contractual Divide.* In a commercial setting, client/supplier relationships are subject to bidding rules and contracts, which often place severe limitations on information exchange between stakeholders and designers. In this chapter, we explore the effect of limitations on the process of optimal quantification of Non-Functional Requirements, and introduce a practice designed to deal with them: Requirements Convergence Planning.

Chapter 4: *How Architects See Non-Functional Requirements: Beware of Modifiability.* This chapter presents the analysis and key findings of a survey about dealing with non-functional requirements (NFRs) among architects.We find that, as long as the architect is aware of the importance of NFRs, they do not adversely affect project success, with one exception: highly business critical modifiability tends to be detrimental to project success, even when the architect is aware of it.

Part II, Establishing a Solution Architecting Approach, starts with a case study on architectural decision making. The following chapters highlight various aspects (RQ-2a-d) that have contributed to RCDA, our solution architecting approach, which is presented in a chapter of its own (RQ-2e):

Chapter 5: *Case Study: Successful Architecture for Short Message Service Center.* This chapter presents and analyzes the key architectural decisions in the design of a successful Short Message Service Center as part of a GSM network, and looks at how architectural choices that deviated from the prevailing "fashion" led to a successful architecture.

Chapter 6: *The Influence of CMMI on Establishing an Architecting Process.* This chapter presents the elicitation of requirements an architecting process needs to address in order to be CMMI compliant. It then analyzes the potential impact of these requirements on generic architecting processes found in literature, and investigates how the CMMI can be extended to better support solution architecting.

Chapter 7: *Successful Architectural Knowledge Sharing: Beware of Emotions.* This chapter presents the analysis and key findings of a survey on architectural knowledge sharing. Impact mechanisms between project size, project success, and architectural knowledge sharing practices and challenges are deduced from the survey's result based on reasoning, experience and literature.

Chapter 8: *Architecting as a Risk- and Cost Management Discipline.* We propose to view architecting as a risk- and cost management discipline. This point of view helps architects identify the key concerns to address in their decision making, by providing a simple, relatively objective way to assess architectural significance. It also helps business stakeholders to align the architect's activities and results with their own goals.

Chapter 9: *Risk- and Cost Driven Architecture: a Pragmatic Solution Architecting Approach.* This chapter describes RCDA, the solution architecting approach developed in Logica. The approach consists of a set of practices, harvested from Logica practitioners and enhanced by the research presented in this thesis. We present the structure of the approach and its rationale, and the result of a survey measuring RCDA's effect among architects trained in the approach.

Chapter 10: *Concluding Remarks.* We present the key contributions of this thesis, draw conclusions and discuss future work.

1.6 Publications

Most of the work in this thesis has been or is about to be published elsewhere:

- Chapter 2 has been adapted from “Resolving Requirement Conflicts through Non-Functional Decomposition,” by Eltjo R. Poort and Peter H. N. de With, pp.145-154, Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA’04), 2004.

- Chapter 3 has been submitted as “Dealing with Non-Functional Requirements across the Contractual Divide,” by Eltjo R. Poort, Andrew Key, Peter H.N. de With and Hans van Vliet to 10th Working IEEE/IFIP Conference on Software Architecture (WICSA’12), 2012.

- Chapter 4 has been adapted from “How Architects See Non-Functional Requirements: Beware of Modifiability,” by Eltjo R. Poort, Nick Martens, Inge van de Weerd and Hans van Vliet, pp.37-51, 18th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ’12), 2012.

- Chapter 5 has been adapted from “Successful Architecture for Short Message Service Center,” by Eltjo R. Poort, Hans Adriaanse, Arie Kuijt, Peter H.N. de With, pp.299-300, Fifth Working IEEE/IFIP Conference on Software Architecture (WICSA’05), 2005.

- Chapter 6 has been adapted from “The Influence of CMMI on Establishing an Architecting Process” by Eltjo R. Poort, Herman Postema, Andrew Key, and Peter H.N.de With, pp.215-230, QoSA’07 Proceedings of the Quality of software architectures 3rd international conference on Software architectures, components, and applications, 2007.

- Chapter 7 has been adapted from “Successful Architectural Knowledge Sharing: Beware of Emotions” by Eltjo R. Poort, Agung Pramono, Michiel Perdeck, Viktor Clerc and Hans van Vliet, pp.130-145, Lecture Notes in Computer Science, 2009, Volume 5581/2009. A version of this paper also appears in [Ali Babar et al., 2009].

- Chapter 8 has been adapted from “Architecting as a Risk- and Cost Management Discipline,” by Eltjo R. Poort and Hans van Vliet, pp.2-11, 2011 Ninth Working IEEE/IFIP Conference on Software Architecture, 2011. An extended version of this paper, including parts of Chapter 9, has been accepted to the Journal of Systems and Software, 2012.

Part I

Dealing with Non-Functional Requirements