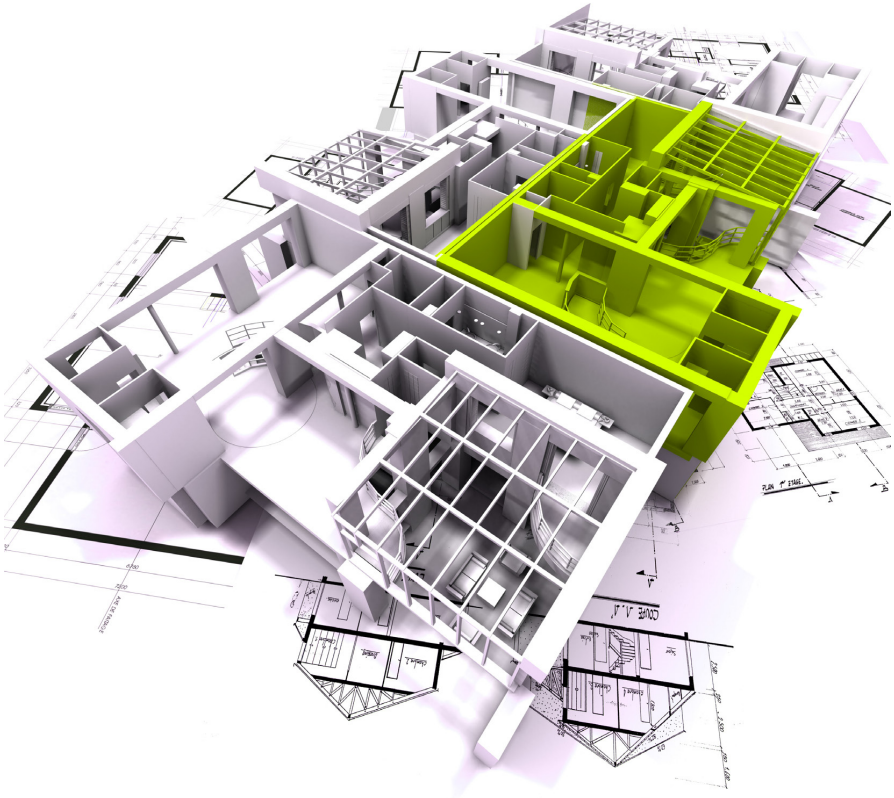
logica
be brilliant together

IMPROVING SOLUTION ARCHITECTING PRACTICES

Eltjo R. Poort



Architecture as a risk- and cost management discipline

Improving Solution Architecting Practices

Eltjo R. Poort

2012

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

The research reported in this thesis has been carried out in cooperation with Logica. Logica is a business and technology service company, employing 41,000 people. It provides business consulting, systems integration and outsourcing to clients around the world.

Promotiecommissie:
prof. dr. P. Kruchten (University of British Columbia)
prof. dr. J.J. Lukkien (Eindhoven University of Technology)
prof. dr. R.J. Wieringa (University of Twente)
dr. P. Lago (VU University Amsterdam)

ISBN 978-94-6191-263-3
NUR 982

SIKS Dissertation series No. 2012-18

Cover illustration by Franck Boston
Typeset in LaTeX by the author

VRIJE UNIVERSITEIT

Improving Solution Architecting Practices

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. L.M. Bouter,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de faculteit der Exacte Wetenschappen
op maandag 18 juni 2012 om 13.45 uur
in de aula van de universiteit,
De Boelelaan 1105

door

Eltjo Rik Poort

geboren te Amsterdam

promotoren: prof.dr. J.C. van Vliet
prof.dr. P.H.N. de With

Contents

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

2

Resolving Requirement Conflicts through Non-Functional Decomposition

A lack of insight into the relationship between (non-) functional requirements and architectural solutions often leads to problems in IT projects. This chapter presents a model that concentrates on the mapping of non-functional requirements onto functional requirements for architecture design. We build a framework that both provides a model and a repeatable method to transform conflicting requirements into a system decomposition. This chapter presents the framework, and discusses two cases onto which the method is applied. In one case, the method is successfully used to reconstruct the high-level structure of a solution from its requirements. The second case is one in which the method was actually used to create a solution design fitting the stakeholders' needs, and that is reproducible from its requirements.

2.1 Introduction

The primary result of any architectural design process is a blueprint of a solution, identifying the main components and their relationships from different views. A topic that is currently under close scrutiny is the derivation of these architectural components from the functional and non-functional solution requirements. The well-known discipline of Functional Decomposition (FD) can be used as a basis, but will by itself rarely yield a solution that fulfills the non-functional requirements. This is not surprising, since rules of Functional Decomposition only deal with generic best practices for achieving software quality, such as high cohesion and low coupling. FD has no rules to deal with solution-specific quality requirements.

Several approaches exist for deriving a solution's architecture from its NFRs:

- [Boehm and In, 1996] identifies a link between NFRs and a set of product and process strategies to address them. The process Boehm proposes to select strategies is called the WinWin spiral model [Boehm and Bose, 1994]; it is basically a negotiation model.
- A group around Lawrence Chung and John Mylopoulos has done extensive work on an NFR Framework, a process-oriented approach to deal with NFRs [Mylopoulos et al., 1992, Chung et al., 1999]. They introduce the concepts of "softgoals" and "satisficing", meaning that goals are set without clear-cut criteria when they are fulfilled. Satisficing is a word for sufficiently satisfying the goals from the stakeholders' point of view. NFRs are modeled as conflicting or synergistic goals in a softgoal interdependency graph. Design alternatives that realize the NFRs can subsequently be evaluated using tradeoff analysis.
- In [Bosch, 2000] the subject is dealt with by first obtaining a functionality-based architecture, and then applying architectural transformations to satisfy the NFRs. A good example of a detailed method using this iterative approach is given in [de Bruin and van Vliet, 2002].
- Publications of the Software Engineering Institute [Bass et al., 2003] show the development of a framework and tooling towards methodical architectural design, based on NFRs: Attribute Driven Design.
- Another group has developed the Component - Bus - System - Property (CBSP) method for iterative architectural refinement of requirements. In [Gruenbacher et al., 2001], the need is mentioned to group artifacts to create an architecture, but no indication is given how to do this.

All the approaches mentioned above rely on knowledge of the effect of a number of known strategies on quality attributes. Every approach needs a pre-existing catalogue of "product strategies" [Boehm and In, 1996], "operationalizations" [Chung et al., 1999], "tactics" [Bass et al., 2003] or similar. The whole Patterns community is based on the need to classify and document such known strategies [Gamma et al., 1995, Buschmann et al., 1996, Gross and Yu, 2001]. In this chapter, we present a more direct approach, based on first principles rather than a catalogue of pre-existing strategies. We have developed a method for decomposing a solution based on the conflicts in the solution requirements. We have named this method Non-Functional Decomposition (NFD) to highlight the contrast with Functional Decomposition, and to emphasize the importance of Non-Functional Requirements in this process.

NFD proposes a method for grouping and splitting of architectural entities based on requirements, and is complementary to the CBSP approach in that sense.

A clear benefit of the NFD approach is that it focuses on non-functional and other supplementary requirements right from the beginning, yielding a defined trace from those requirements to the solution structure. Moreover, the development process and its requirements are also integrated in the approach, giving a better basis for architectural and project decision trade-offs.

The sequence of this chapter is as follows. First, we will present and discuss some of the shortcomings of the generally accepted model for the architectural design process. We will then develop a refined model of this process. Then we will describe the process for deriving solution structure from supplementary requirements that is based on this model. The succeeding sections then describe two cases: one in which the NFD method was used, and one in which it is applied retrospectively to show its validity. We conclude with a discussion.

2.2 Motivation for Non-Functional Decomposition

Our interest in Non-Functional Decomposition is based on a number of distinct observations from our substantial experience in architectural design.

- *Cohesive force of supplementary requirements:* good architectures tend to cluster functions with similar supplementary requirements in the same subsystem.

- *Divide-and-conquer conflict resolution principle:* if a subsystem has to fulfill conflicting requirements, it is useful to separate the parts that cause the conflict(s).

- *Entanglement of function, structure and building process of software:* these three elements are highly interrelated.

NFD is a framework consisting of both a model of the elements involved in the architectural process, and a method for architecting software-intensive systems based on solution requirements. It is a framework in the sense that it does not venture into the details of achieving specific quality attributes (or other supplementary requirements); there is ample literature available for each conceivable attribute. Rather, it highlights the relationships between these requirements, their conflicts and ways to resolve them. It also helps in making choices about the development process.

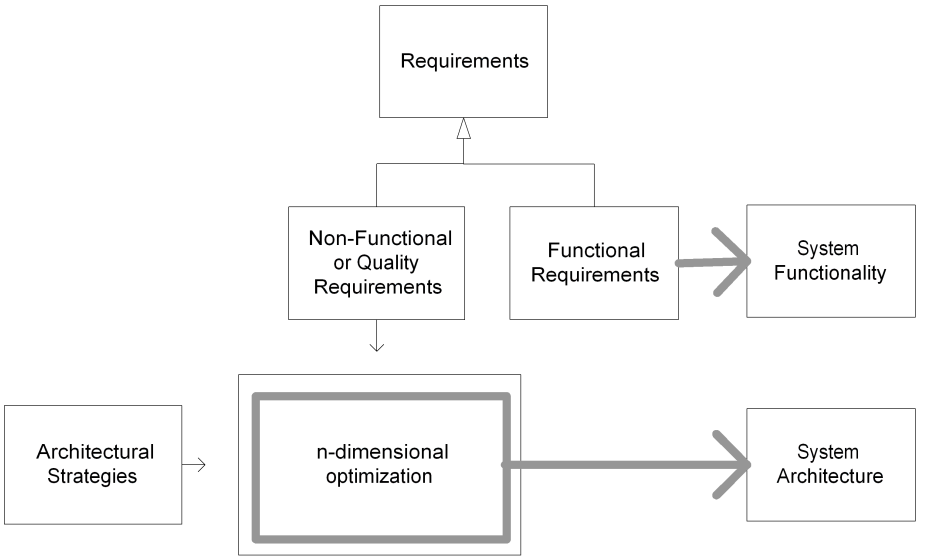


Figure 2.1: Accepted model of relationship between requirements and architecture.

2.3 Model of Requirements and Architecture

2.3.1 Accepted model for architectural design

When studying the available literature on the relationship between requirements and architecture (see §2.1), the following widely accepted model emerges.

System Requirements are usually divided into Functional and Non-Functional Requirements. These Non-Functional Requirements (NFRs) are often referred to as Quality (Attribute) Requirements; these two are treated more or less as synonyms. A generally accepted principle is *the leading NFR principle*: in designing system architectures, the Non-Functional or Quality Requirements are at least as important as the Functional Requirements. In order to satisfy NFRs the software architect applies Architectural Strategies to the system design, such as design patterns, layering techniques, etc. The architect's task then becomes an n-dimensional optimization problem: find the combination of architectural strategies that yields a solution with the best fit to the n NFRs.

The implicit model underlying this reasoning is depicted in Fig. 2.1. Although the simplicity of this view has its merits, in our experience it has some shortcomings. Par-

ticularly, the relationship between quality attributes and non-functional requirements is oversimplified, and it ignores the fact that functional requirements can also be very important in architectural design. It also ignores that NFRs often put constraints on the solution development *process* rather than on the solution architecture, implying that architectural choices are not the only contributors to satisfy NFRs. Conversely, requirements on the development process like project deadlines and budget limitations can have a large impact on solution architecture.

2.3.2 Refined requirements classification for NFD

Our new NFD model, as is illustrated in Fig. 2.2, refines the classification of requirements, and is more detailed on the non-functional aspects. Two major differences come to the foreground: functional requirements are split into primary and secondary functional requirements, and the secondary functional requirements are grouped together with the non-functional requirements. This group is called *supplementary requirements*. Additionally, a distinction is made between two types of non-functional requirements: quality attributes and delivery requirements.

Let us now define the Primary and Supplementary Requirements groups in more detail.

Primary Functional Requirements are demands that require functions which directly contribute to the goal of the solution, or yield direct value to its users. They represent the principal functionality of the solution. The identification of primary requirements (which ones to select) is similar to determining which processes in an organization are primary processes. All primary requirements are functional (there are no non-functional primary requirements), but not all functional requirements are primary requirements, as will be explained in the next section.

Supplementary Requirements represent all other requirements imposed on the solution. They can be functional or non-functional. Supplementary requirements (SRs) are always *about* primary requirements, and usually put constraints on how the primary functionality is implemented. In the NFD model, the Supplementary Requirements are further divided into three subcategories:

1. *Secondary Functional Requirements* (SFRs) require functionality that is secondary to the goal of the solution. Examples are functions needed to manage the system or its data, logging or tracing functions, or functions that implement some legal requirement. Like all other SRs, they usually apply to a particular subset of the primary requirements. For example, “All transactions in module X should be logged”, or “access to data in table Y is subject to authorization

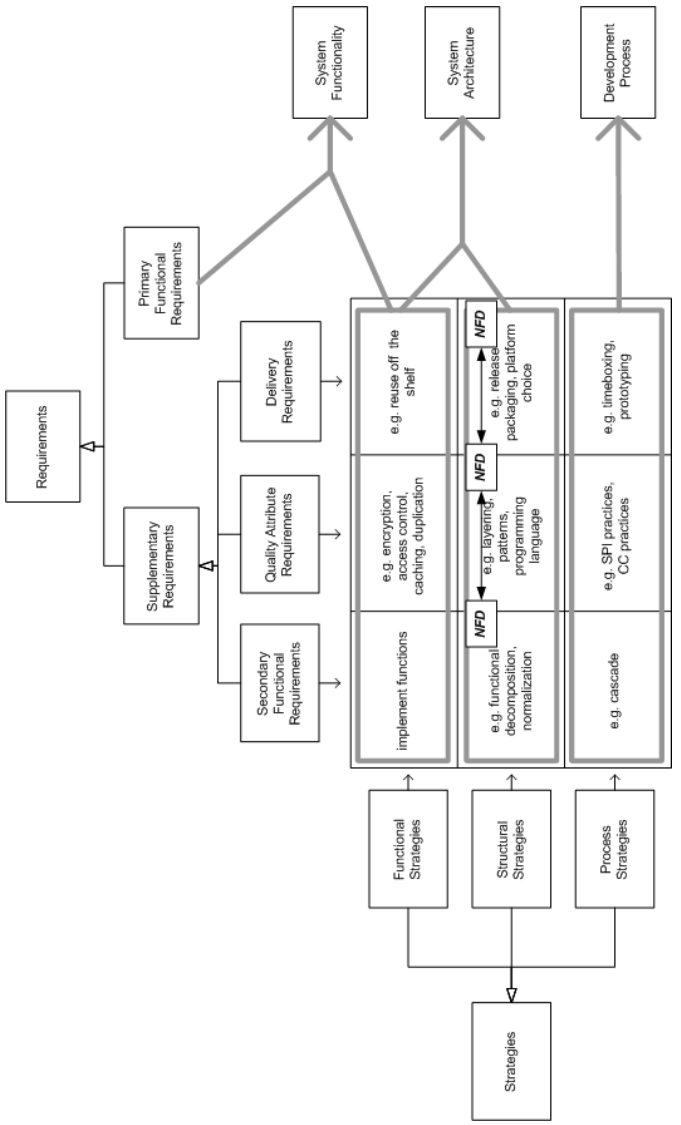


Figure 2.2: The NFD model of the relationship between solution requirements and architecture.

according to model Z". SFRs are usually not quantifiable: the solution either has the functionality or it doesn't.

2. *Quality Attribute Requirements* (QARs) are quantifiable requirements about solution quality attributes. They can always be expressed as a number and a scale, e.g. following Gilb's notation techniques [Gilb, 2005][1]. Examples of QARs are reliability, usability, performance and supportability. There are many taxonomies available, e.g. SQuaRE [ISO/IEC 25000, 2005].

3. *Delivery Requirements* constitute the third category of supplementary requirements. They put constraints on the solution that cannot be measured by system assessment, and incorporate e.g. managerial issues. Examples of DRs are time-to-market, maximum cost, resource availability and outsourcibility. Delivery requirements can be expressed in "-ilities" that make them resemble quality attribute requirements, such as affordability or viability, but they are not about solution quality. However, they can be just as important to solution design as functional or quality requirements.

A solution's compliance with QARs and SFRs can in principle be measured by anyone having access to the system once it has been realized, *regardless of whether they know about its history or its cost*. Compliance with Delivery Requirements can only be assessed by looking at how the solution was realized.

There is a relationship between Secondary Functional Requirements (SFRs) and functional solutions to Quality Attribute Requirements, which will be discussed later. SFRs can usually be traced back to a high-level quality need, but to express them as a quality requirement would leave too much room for interpretation. For example, the requirement to log system errors over an SMTP interface is an implementation of a manageability need, but to just require that "System management should require at most 0.1 FTE" would allow other, perhaps less desirable solutions. Satisfying Quality Attribute Requirements may also entail adding functionality to the solution, but this time the choice of functionality is at the architect's decision.

2.3.3 The nature of requirement conflicts

The reason for a classification into primary and secondary requirements is a preparation for the NFD process that leads to solution decomposition exploiting the requirement conflicts. The NFD version of the leading NFR principle cited above is that *in designing solution architectures, the supplementary requirements are more important than*

[1] Tom Gilb first introduced these techniques in [Gilb, 1988], and later incorporated them in the "planguage" notation [Gilb, 2005]

the primary requirements. Primary requirements are never conflicting: if they would, the requirements would be intrinsically inconsistent or the problem statement poorly posed. However, supplementary requirements including secondary functional requirements, can appear to be conflicting, as is explained in the following paragraph.

Requirements on a software system are not intrinsically conflicting, because conflicts arise from limitations in the design strategy domain. Boehm and In have based their software tools for identifying quality-requirement conflicts on this fact [Boehm and In, 1996]. We have further analyzed the common strategies to satisfy supplementary requirements, including quality attributes. Our analysis clarifies that some quality attributes and delivery requirements may be so tightly bound to certain types of strategies, that they are effectively inherently conflicting. This situation arises when a quality attribute can only be achieved by one class of strategies, and when this class of strategies is invariably detrimental to another quality attribute. The Feature-Solution graphs introduced in [de Bruin and van Vliet, 2002] provide a good way to visualize these conflicts. We will illustrate this point with a few examples.

We have categorized the strategies for fulfilling software quality requirements into three types and nicknamed them the three strategy dimensions of solution construction: the *process* dimension, the *structure* dimension and the *functional* dimension.

1. One way to achieve supplementary requirements is by making choices in the software building *process*. Models like the Capability Maturity Model Integration [CMMI Product Team, 2010] and other software process improvement practices generally aim at improving the quality of software. Recommended practices have been documented to achieve certain quantified Safety Integrity Levels [IEC 61508, 1999] or to fulfill certain security requirements [CCPSO, 1999]. These practices tend to make the software construction process more expensive, giving rise to the first example of inherently conflicting requirements: *reliability* versus *affordability* (not an NFD quality attribute, but possibly a delivery requirement).

2. Another way to influence quality attributes or to satisfy other supplementary requirements is by making choices in the *structure* of the software. Examples of software structuring solutions include layering, applying of design patterns, choosing higher or lower level languages, modifying the granularity or modularity of the software, and so on. We have started to explore this area somewhat in [Poort and de With, 2003]. Generally speaking, the structure-based solutions seem to have one common element: more structure (i.e. higher level programming language, more layers, higher granularity etc.) means better modifiability, but less efficient code. This is the second example of inherently conflicting quality attributes: *modifiability* versus *efficiency*.

3. The third way to achieve supplementary requirements is by building *functionality* that is specifically aimed at achieving a quality or delivery objective. Examples are encryption and access control functionality to achieve a certain security goal [CCPSO, 1999], or caching functionality to achieve a certain response time and thus increase usability. Although these types of strategies are not specifically detrimental to other quality attributes, they do increase the size and complexity of the solution, leading to effects such as lower affordability and reliability. The reader should note the difference between secondary functional requirements with underlying quality needs and functional strategies aimed at satisfying quality attribute requirements. In the former case, the functional strategy is raised to the status of requirement and the responsibility of the requirement specifier. In the latter case, the functional strategy is the responsibility of the architect. In practical situations, this dichotomy may be ambiguous and quality needs are translated into functional solutions by an iterative process that involves both the requirements specifier and the architect.

In the above examples of "inherently conflicting" requirements, the conflicts emerge when applying the solution strategies to a single subsystem or component. These conflicts can often be resolved by separating the subsystem or component into different parts, and applying different solution strategies to the respective parts. Viewed from this perspective, modifiability and efficiency need not be conflicting: one can decompose a solution into low-coupled subsystems for modifiability, and then apply strategies for making the code of each subsystem more efficient. Approaches of this kind are put into practice intuitively by experienced architects, and we have modelled them in our Non-functional Decomposition Framework.

2.3.4 Applying solution strategies

The NFD model of the architecture process refines the n-dimensional optimization problem of the consensus model into a 3×3 matrix. The cells of this matrix contain strategies from each of the three strategy dimensions fulfilling each of the three types of requirements. Whereas the diagonal of the matrix contains obvious strategies (e.g. functional solutions to functional requirements), the off-diagonal cells often suggest important solutions that can help achieve requirements that would otherwise pose problems. Without being complete, we provide some examples of each of the matrix cells.

Functional strategies aimed at functional requirements: the required functions should be implemented.

Functional strategies aimed at quality-attribute requirements: these are functions

like encryption, access control, caching and duplication, that are specifically designed to achieve a quality objective.

Functional strategies aimed at delivery requirements: delivery requirements, like outsourcing or time-to-market limitations, often exist. Their realization points to e.g. reuse of off-the-shelf components or purchasing and integrating commercial products.

Structural strategies aimed at functional requirements: examples of structures that contribute to functional requirements are database normalization, extracting of generic functionality, functional or non-functional decomposition.

Structural strategies aimed at quality-attribute requirements: lead to programming in patterns, but there are also other structural strategies contributing to quality attributes, such as the choice of programming language or correct parametrization. The NFD method itself also contributes to fulfilling quality-attribute requirements.

Structural strategies aimed at delivery requirements: delivery requirements like preferred release schedules can be realized by adapting the structure of the solution to accommodate incremental deployment. Another example is the choice of a rapid development platform (fourth generation language), which dictates a particular solution structure. NFD can also be applied here.

Process strategies aimed at functional requirements: an example is using a conventional cascade-development method, which prioritizes system functionality over time and budget limitations.

Process strategies aimed at quality-attribute requirements: examples of these are best practices from the Software Process Improvement community to improve reliability, or Common Criteria assurance packages to achieve security goals.

Process strategies aimed at delivery requirements: delivery requirements such as "user involvement" can be realized by prototyping, or "strict deployment deadline" by using a development method such as EVO [Gilb, 2005] or the Rational Unified Process®[2] (RUP®) [Kruchten, 1998] that employs time-boxing techniques.

The combination of applied strategies in the process dimension results in the best *development process* to fit the solution requirements. The sum of the applied functional strategies and the realization of the primary functional requirements together compose the *solution functionality*. The *solution architecture* consists of a high-level description of the applied functional (logical view) and structural (subsystem, development, deployment views) strategies.

[2]RUP, Rational and Rational Unified Process are trademarks of International Business Machines Corporation.

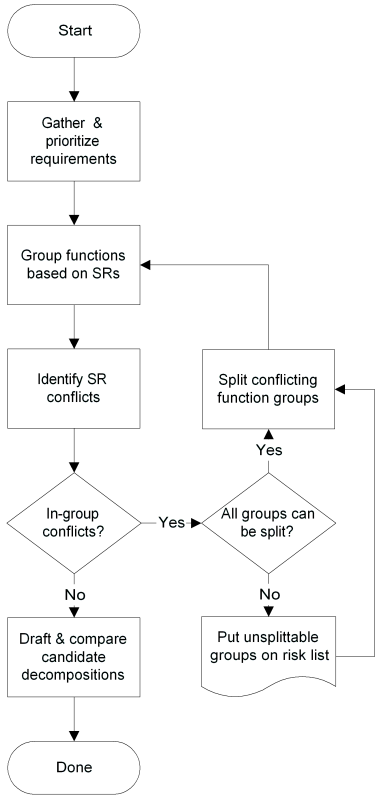


Figure 2.3: The NFD Process.

2.3.5 The role of the NFD process

Non-Functional Decomposition (NFD) is a strategy in the structural dimension of solution construction. The NFD process helps to optimize the structure of the solution for all supplementary requirements, including delivery and secondary functional requirements, which are generally associated with process or functional strategies first. It does this by adapting the solution structure to the requirement conflicts in the solution, and isolating conflicting requirements in subsystems that can then be individually optimized by applying process, structural or functional strategies. It is essentially an iterative divide-and-conquer strategy for resolving requirement conflicts.

2.4 The NFD Process

The process of NFD is depicted in Fig. 2.3 and contains the following steps.

Gather and prioritize requirements can be based on any modern requirements elicitation technique, provided that the documented requirements show how the Primary Functional Requirements (PFRs) are mapped to the Supplementary Requirements (SRs). It is important that the requirements are somehow prioritized, since prioritization of PFRs is important for project and release planning, and prioritization of SRs is important for the architecture. *Example from the Unified Process:* in the UP, FRs are generally documented as use cases, and SRs as supplementary specifications. The NFD method requires that the supplementary specifications are made specific to (groups of) use cases, e.g. by documenting them in the Use-Case Descriptions, or specifying to which use cases SRs apply in the Supplementary Specifications document. Another way of linking SRs to FRs is the use of quality attribute scenarios as described in [Bass et al., 2003].

Group functions based on supplementary requirements is the process of finding all (primary) functional requirements that share or have similar supplementary requirements, and grouping them together. This will yield a number of cross-sections of the functionality of the solution, depending on which supplementary requirement is used as a grouping criterion. In this step, the distinction between PFRs and SRs is less important: each group will have a number of functions, originating from both primary and secondary requirements, which will be treated equally during the remainder of the process. *Example:* a time-to-market priority grouping will divide functionality into groups that are candidates to be included in different release phases of the solution, while availability grouping will divide functionality into candidate groups to run on platforms with differing availability characteristics.

Identify supplementary requirement conflicts yields two types of conflicts:

1. *Grouping conflicts* are caused by differences in grouping of functions, i.e. the grouping of the functions is significantly different from one SR to the other. *Example:* there are three function groups, called WorkFlow, DataEntry and Analysis (Fig. 2.4). Security requirements for DataEntry and Analysis are similar and more restrictive then those for WorkFlow, but modifiability requirements for Analysis are more stringent than those for DataEntry and WorkFlow.

2. *In-group conflicts* are conflicting supplementary requirements within one function group. *Example:* the Analysis function group from the previous example has both critical performance requirements and high modifiability requirements.

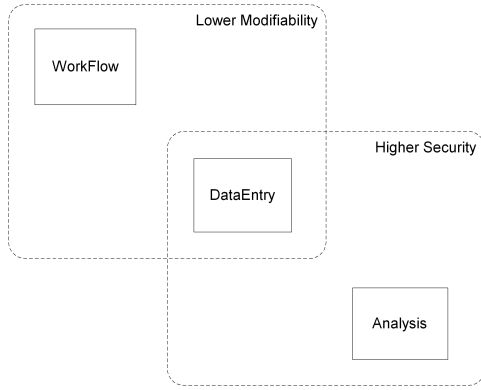


Figure 2.4: Grouping conflict example.

Split conflicting function groups deals with in-group conflicts. Most of the time, a further analysis of an in-group conflict will show that the conflicting requirements can actually be assigned to different functions. These functions are then separated, leading to a splitting of the function group. The resulting two or more new function groups may then be reconsidered for being included in other function groups, so the process re-enters the "Group functions based on SRs" stage. This loop is repeated until no in-group conflicts are left that can be split further. Function groups that cannot be split in any way are flagged as risk factors. They deserve close attention during the rest of the process and need to be dealt with prior to large-scale project implementation, e.g. in an architectural prototype.

Draft and compare candidate decompositions: After the in-group conflicts are solved, the resulting grouping conflicts will be the basis for the architectural decomposition. A number of candidate decompositions into architectural components result, each favoring the main supplementary requirement that the function grouping is based upon.

At this stage, prioritization of supplementary requirements becomes important. In our experience, the candidate decomposition that is based on the SRs having the highest stakeholder priority, yields the architecture that best fits the stakeholder requirements. This does not mean that we suggest that the n-dimensional optimization problem mentioned earlier can be reduced to a series of one-dimensional optimizations, designing for the most important requirement first and then narrowing down the design choices further for each requirement. But in the decomposition process, it turns out that the decompositions based on the SRs with the highest priority have the best chance of yield-

ing a solution the stakeholders can live with. In case of doubt, an impact analysis of the top three decompositions can be made, e.g. by using the CBAM method [Kazman et al., 2002]. The decomposition with the best fit to the stakeholder's needs (including risk and cost) is selected for implementation.

2.5 Case Study: Criminal Investigation System

2.5.1 Background

Two IT organizations affiliated with the Dutch ministry of internal affairs, the Concern Informatiemanagement Politie (CIP, "Concern Information Management Police") and the ICT-Service Coperatie Politie, Justitie en Veiligheid (ISC, "ICT Service Cooperation Police, Justice and Safety"), were developing a product line for nationwide processing of and access to criminal investigation and intelligence data.[3] The product line was called Politiesuite Opsporing (PSO, "Police Suite for Investigation"). One of the authors was the lead software architect for PSO, and NFD was used to design the suite's top-level decomposition. The main challenge was to create an architecture that would allow the addition of many new products to the suite in the years to come, without compromising the strict privacy and confidentiality requirements on the system.

2.5.2 Summary of requirements

The suite's primary functionality is the support of all business processes related to criminal investigation, including management of the processes, gathering of data through multiple channels, and structuring and analysis of the data. The three most important supplementary requirements according to the stakeholders are:

SR1 *Authorization:* access to criminal investigation data is restricted by special privacy laws. Unauthorized access to privileged data is by far the biggest threat to a criminal investigation system.

SR2 *Reliability:* reliable application of authorization and other business rules is crucial. The system should be designed in such a way that the enforcement of especially authorization rules is reliable and stable, even after several product generations.

[3]The authors would like to thank the ISC and CIP organizations for granting permission to publish this case study.

SR3 *Development time:* the criminal investigation systems currently in use are based on obsolete architectures and there is an urgent need in the field to support new functionality. Exceeding the stated deadline of one year of development time is unacceptable.

SR1 is a secondary functional requirement, SR2 a quality attribute requirement, and SR3 a delivery requirement.

2.5.3 Results

NFD was applied by first mapping the most important supplementary requirements onto the functional features, and then basing the main architectural decomposition on this mapping. SR1 applies specifically to the data gathered for criminal investigation purposes. It turned out that the most reliable and best maintainable solution for the future was to create a central component for access control and storage of these data. Since the legal name for storage of such data is a police register, this central component was named the "register vault". By using off-the-shelf components supplied by a database vendor, the register vault could be assembled and an architectural prototype evaluated within a few months time, making a good start at satisfying SR3.

In this case, RUP was used to streamline the development process. The RUP Supplementary Specifications artifact is the place to document quality and other supplementary specifications, but the standard template treats these as system wide or "general" requirements. We changed the template slightly to accommodate documenting the mapping between primary and supplementary requirements. We did this at the level of "features" as defined by RUP. This allowed us to document the trace from primary and supplementary requirements to the system decomposition design decisions.

In the end, the Non-Functional Decomposition principles turned out to be very useful in communicating to the stakeholders how our design decisions were related to their stated supplementary requirements.

2.6 Case Study: Dutch Road-Pricing System

In this section we will apply the NFD model and process to analyze a large system on roadpricing. One of the authors was involved in this project (until it was suspended for political reasons), which is described below. Although NFD was not available at the time, applying the method retrospectively to this case presents a good illustration of its principles.

2.6.1 Background

In the late 1990s, the Dutch government decided to drastically change the tax system for automobile owners and drivers. Automobile tax traditionally consisted of gasoline tax, annual motor vehicle tax and BPM (personal motor vehicle tax payed once when purchasing a vehicle). The system should be replaced by a direct tax system based on usage: gasoline tax would be reduced to the legal European minimum, and annual motor vehicle tax and BPM would be completely replaced by a roadpricing scheme called "Kilometerheffing" ("Charging by kilometer"), hereafter referred to as KMH. By differentiated pricing of road segments based on the time of day and location, the scheme could also be used to make drivers avoid congested (and thus more expensive) areas during rush hours. Feasibility of the scheme would highly depend on the use of IT systems, some of which would have to be in the vehicle. The government planned to share the cost of developing and manufacturing the in-vehicle systems (called "Mobimeters") with the multimedia, communication and automobile industries by making generic components of those systems available to those industries.

2.6.2 System requirements

The Mobimeter requirements were shared with industry partners[4] in order to facilitate discussion. Without going into too much detail, the original requirements can roughly be summarized as follows.

PF1 The system shall continuously measure the position and driving direction of the vehicle.

PF2 The tariff used for calculating the cost during a journey is determined on the basis of the following parameters (hereinafter referred to as Tariff Parameters):

- Date and time;
- Vehicle position;
- Direction of travel;
- Tariff table;
- Vehicle category.

PF3 A driver shall be notified of the applied tariff while driving.

PF4 The system shall determine the distance travelled by a vehicle.

[4]The original requirements were drawn up by a team led by Maarten Boasson (University of Amsterdam). They were based on the "Mobimiles" report of Roel Pieper (University of Twente), which was never formally published

PF5 The mobility costs due is calculated as being the product of the distance travelled times the current tariff.

PF6 At least once every month in which 1000 kilometers has been driven or at least once per elapsed year, whichever comes earlier, all data shall be communicated to the tax office.

PF7 The system shall be able to receive new tariff tables.

The associated supplementary requirements (SRs) are summarized below. For brevity, we only mention the most important ones:

S1 *Privacy:* a vehicle's mobility patterns may not be deducible, either in real time (tracking) or afterwards from system data (tracing).

S2 *Verifiability:* the KMH Road-Pricing System shall enable verification that the road pricing charge has been determined correctly, without requiring more than one physical inspection per year.

S3 *Provability:* The system shall enable drivers to verify the correctness of the charges by inspecting all relevant data.

S4 *Security:* All data required for the KMH Road-Pricing System process will be protected against unauthorized modification.

S8 *Re-usability:* all in-vehicle system functions that could be useful for other applications shall be made available for re-use by third parties.

S12 *Viability:* industry partners and the relevant governmental and non-governmental organizations shall be involved as much as possible in the development of the KMH system.

S13 *Standardization:* any interfaces to be designed for in-vehicle equipment shall be developed in close cooperation with the relevant standardization bodies.

S1, S4 and S8 are quality-attribute requirements. S2 and S3 are secondary functional requirements. S12 and S13 are delivery requirements that originally did not occur in the requirements document, but in the project plan. We mention them here because in the NFD model they qualify as supplementary requirements. As will be shown, they did impact the system architecture.

Table 2.1 shows how the original supplementary requirements map onto the primary requirements. Note that some of the mappings apply to subsets of a particular

Table 2.1: Original mapping of supplementary onto primary requirements.

	PF1	PF2	PF3	PF4	PF5	PF6	PF7
S1	X					X	
S2	X	X		X	X	X	
S3	X	X		X	X	X	
S4	X	X		X	X	X	X
S8	X		X			X	X
S12	X		X			X	X
S13						X	X

primary functions only, e.g. standardization of *only* the interface, or protection of *only* the data:[5] a finer granularity of functionality will help us split and re-group the functions later on. The next step in the mapping process would be to split up the PFs to achieve a more exact mapping, but that exercise would be too detailed for this thesis.

A number of grouping strategies present themselves, but let us first look at the glaring in-group conflict concerning privacy versus verifiability and provability in the group PF1+PF6. According to NFD, we split the group to resolve the conflict. Clearly, if all data on which the charge is based are communicated to the tax office, the privacy requirement is violated. The objective is to split PF6 in such a way that only less privacy-sensitive data are communicated and simultaneously maintain the verifiability. After checking back with the stakeholders for the business-need behind PF6, it turns out that we can split as follows

F6a At least once every month in which 1000 kilometers has been driven or at least once per elapsed year, whichever comes earlier, the total charges and the total distance per tariff shall be communicated to the tax office.

F6b Spot checks: a travelling vehicle shall be able to answer challenges made by roadside-enforcement equipment by transferring all currently measured data and the current tariff table.

F6c On request by the driver, the system shall communicate all data used to calculate charges to him or her.

F6a is sufficient to fulfill PF6's underlying need; F6b is a functional solution to S2, and F6c is a functional solution to S3. F6b adds a short-range communication function

[5] The presence of supplementary requirements that apply to data elements or storage is quite common in our experience; these requirements often lead to special data storage components, especially if they have high priority.

Table 2.2: Second iteration mapping of supplementary onto primary requirements.

	PF1	PF2	PF3	PF4	PF5	F6a	F6b	F6c	PF7
S1	X						X	X	
S2	X	X		X	X	X	X		
S3	X	X		X	X	X		X	
S4	X	X		X	X	X	X	X	X
S8	X		X			X	X	X	X
S12	X		X			X	X	X	X
S13						X	X	X	X

to the Mobimeter, which falls under the re-usability and standardization requirements S8, S12 and S13. This analysis leads to a new mapping table.

Table 2.2 shows the SR/FR mapping after splitting PF6. The conflict between S1 and S2 is now isolated in component F6b, the spot check function. This reflects the issue that privacy-sensitive data are present in the spot-check equipment. We put this conflict aside as a risk that will be managed by a protocol surrounding the management of these data: how long they may be stored, to what purpose, etc.

Let us now look at grouping criteria. According to the stakeholders, security, privacy and viability through re-usability are the most important supplementary requirements and in that order of priority. The security requirement S4 groups the associated data of all functions except the driver display, which basically means that the Mobimeter should contain a secure data storage component or "trusted element" that all KMH functions should have access to. Privacy requirement S1 no longer applies to F6a, since we split off the data from which the mobility pattern can be deduced. So S1 now groups PF1, F6b and F6c. S1/F6b becomes the basis for storage requirements on the roadside spot-check system, and the PF1/F6c group leads to a subsystem called the "user log".

Finally, the viability and re-usability requirements S8 and S12 group the communications functions of F6a/b/c and PF7, the display of PF3 and the vehicle localization of PF1 into a subsystem called the "In-vehicle Telematics Services platform". This platform is designed to have standardized interfaces (S13) both to the KMH-specific part of the in-vehicle equipment and to any additional (optional) service-related components such as a navigation system. It gives access to services like GPS localization and long-range and short-range digital communication. The final system decomposition is shown in Fig. 2.5. In the final version, the localization function was separated from the ITS platform in order to fulfill an additional reusability requirement on GPS equipment, which was already supplied in some cars.

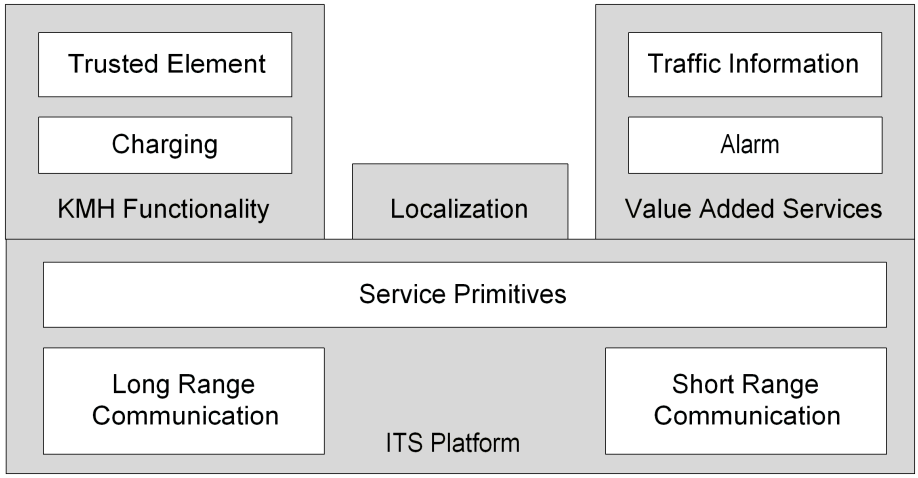


Figure 2.5: Mobimeter architecture.

2.7 Conclusions and Discussion

In this chapter, we have presented the Non-Functional Decomposition (NFD) model as a technique to bring more clarity and structure in the mapping of requirements onto a solution architecture. The key of our technique is to split the requirements into primary and supplementary requirements, and to create a mapping between those categories. The NFD process helps in optimizing the structure of the solution for all supplementary requirements, including delivery and secondary functional requirements. NFD adapts the solution structure to the requirement conflicts in the solution and isolates conflicting requirements in subsystems that can then be individually optimized by applying process, structural or functional solution strategies of which examples were presented.

The PSO product line was presented as a case study of the application of NFD. The main result here was a well documented traceability between supplementary requirements and solution decomposition design decisions. This traceability supported the project team in communicating to the stakeholders the effects of their stated requirements, and the rationale behind the main design decisions.

The KMH project was used as another case study, where we have indicated how the Mobimeter architecture as it was published, can be reconstructed with the NFD model. The examples of resolving in-group conflicts and grouping functions according to supplementary requirements were given to show the application of the method and

principles of NFD.

The validity of the observations on architecting is not only confirmed by our daily work, but can be easily verified by evaluating successful architectures like client/server or n-tier architectures. The components in these architectures all differ in their supplementary behavior, and display specific geographical accessability, modifiability, efficiency or portability attributes.

Of the existing requirements engineering literature, a large part focuses mainly on obtaining and maintaining the right requirements [Robertson and Robertson, 2006, Wiegers, 2003, Jackson, 2001, van Lamsweerde, 2009]. Decomposition is important in these approaches, but applies to the structure of the requirements, rather than the structure of the solution. In §2.1, we list a number of existing approaches in literature for deriving a solution's architecture from its NFRs[Boehm and In, 1996, Bosch, 2000, Chung et al., 1999, Bass et al., 2003, Gruenbacher et al., 2001]. As mentioned before, these approaches all rely on a pre-existing catalog of strategies, called by various names. NFD adds to these approaches a common rationale behind the strategies: a rationale based on the principle that conflicting NFRs can be dealt with by separating the functions that they apply to in the solution structure.

We view NFD as a framework that uses the observations made in §2.2 to improve the architecting process. These observations are not new, we believe they have always been implicit in the work of experienced designers and existing patterns and tactics. By making them explicit, NFD makes the architecting process of transforming requirements into solution design more reproducible, more transparent and more reliable. It also reveals rationale behind existing architectural patterns and tactics, and can be helpful in developing new patterns and tactics to deal with conflicting NFRs. Future work is in further application of NFD in actual technically complex projects, and in the exploration of other areas in which it could be deployed.

3

Dealing with Non-Functional Requirements across the Contractual Divide

Agreement on Non-Functional Requirements between customer and supplier is crucial to a successful IT solution delivery project. In an ideal world, stakeholders and designers cooperate to achieve their common goals in a win-win situation. In a commercial setting, however, one dominant feature often introduces powerful forces from outside the technical realm. That feature is the customer/supplier relationship, usually formalized in bidding rules or as a delivery contract. Formal customer/supplier relationships 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 explore a number of avenues to deal with them.

3.1 Introduction

In the commercial setting of bespoke system development and integration projects, customers and suppliers embark on a quest to converge to a point where requirements can be agreed between them. Starting points on this quest are the customer's needs and the supplier's capabilities to meet those needs. Agreeing on functional requirements is usually the first step. The harder part is agreeing on non-functional requirements, which are more tightly tied to the architecture. One of the aspects that get in the way of a proper integrated approach to develop NFRs and Architecture is the formal character of the client/supplier relationship. This formal character is expressed in bidding and

tendering rules in the pre-contract stage, and in the contract itself after that. The formal representation of NFRs in these situations is often a number, especially when they refer to solution quality attributes: a quantified NFR. The process of getting to these numbers is called NFR quantification.

3.1.1 Requirements and architecture in client/supplier situations

Traditionally, designing a solution to fit stakeholders' needs is done in two phases:

RE : *Requirements Engineering* expresses the needs of the stakeholders in a set of Requirements (FRs and NFRs).

AD : *Architectural Design* finds the optimal solution to address the Requirements, and expresses the solution in a Solution Architecture.

In the commercial setting of fixed price IT projects, Requirements Engineering is done by the customer, and Architectural Design by the supplier. After RE, the customer invites a number of potential suppliers to bid for the privilege of supplying a solution that fulfills the Requirements. This invitation is usually called Request for Proposal (RfP) or Invitation to Tender (ItT); we will use the term RfP. After receiving the RfP, the candidate suppliers will perform enough of the Architectural Design to be able to calculate the cost and time needed to deliver the Solution within a reasonable margin of error.

As stated in Chapter 1, 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]. In other words: of all the Requirements in an RfP, the NFRs have the biggest role in the Architectural Design. In Chapter 2, we have discussed various existing approaches to derive Architectural Design from NFRs, and we presented our own approach to do the same. However, as already noted by [Boehm and Bose, 1994], the notion that an architecture can be derived from requirements in one go is an oversimplification. Architecture and requirements are so closely related, that many aspects of requirements engineering can only be addressed properly if the architecture is developed at the same time. This point is made particularly eloquently in [Paech et al., 2002], which pleads for a tightly integrated approach for Functional Requirements, Non-Functional Requirements and Architecture. In our experience such an integrated approach is indeed necessary, but it is particularly difficult to achieve in the type of fixed price tendering situation described above. This is due to the traditional strict separation of roles in the tendering process, a separation that is mandated by law [US

Government, 2005, European Commission, 2004] for many government related organizations.

In this chapter, we will first present some real-life examples of the issues related to NFR quantification in a commercial setting. We will then look at the problem of NFR quantification from a number of perspectives. We will see how the tendering process interferes with proper NFR quantification, and discuss ways of dealing with this interference.

3.2 Real-life Issues Dealing with NFRs

Sometimes NFRs are explicitly tied to RfPs or delivery contracts; sometimes they are only implicitly mentioned, or ignored altogether. Either way, their impact is significant in all but the smallest projects. Conflicts between customer and suppliers can often be traced back to NFRs. Even in conflicts that revolve around the delivery of certain functionality, the delivery is often delayed by performance or security requirements related to that functionality.

This section presents some typical anonymous real-life experiences and dilemmas. The cases are used for illustration in subsequent sections.

3.2.1 Case #1: the difficulty of communicating NFRs

A customer in the financial services market invites tenders to deliver a bespoke solution to support a funds collection process. NFRs are not stated by the customer in the tender requirements documentation, and are commensurately not addressed by the (subsequently winning) supplier. At the outset of the project, an experienced engineer emphasizes to the project manager the importance of formally agreeing performance with the customer, for fear of otherwise failing to secure acceptance. The project manger agrees and sanctions work to commence on defining a non-functional requirements specification that the customer will be asked to approve.

The engineer and his team spend a significant amount of time producing a specification of performance requirements for each of the functional transactions in the system. The specification, due to the difficulties of precisely specifying such requirements, turns out to be large, highly technical and difficult to understand. The document is presented to the customer for approval, who predictably does not understand it, and engages an independent consultant to help him. The consultant spends a considerable time digesting the specification and the value of the requirements to his client, by which time the delivery project is well advanced and the project manager is beginning to panic. He convenes a meeting with the customer and his consultant to try to force

agreement of the requirements, which fails as the consultant has advised his client that the requirements as stated in the specifications are highly artificial, overly qualified and of little real value. A protracted series of subsequent meetings fails to achieve agreement and meanwhile the main development is nearing completion. The impasse continues to the point that the project starts to incur significant financial loss.

3.2.2 Case #2: the unexpected cost of high availability

A bid team is responding to an invitation to tender a solution for providing administrative support to the provision of key public services. The customer states that the solution must be highly available, stating 99.999% availability requirements with onerous penalties on breach of the operational requirement. Considering the requested services levels, the bid team embarks upon a process of crafting a solution architecture with extreme high availability qualities throughout. The tender process allows little contact with the customer to refine understanding of the customer's stated requirements.

As the bid proceeds and the solution unfolds, it becomes evident that the apparently necessary hardware and software infrastructure costs are very significant. However, the bid team is convinced that this is justified and that all competitors in the tender will be responding similarly. Costs continue to spiral as the impact of the highly available solution on envisaged transactional performance becomes apparent and ever more capable infrastructure is included to compensate.

In the later stages of the bid process and pre-contract award, the bid team is invited to present the solution to the customer: the customer is aghast at the elaborate nature of the solution and the likely price to deliver it. The customer decides to suspend the tender having not received any affordable solution. The customer and all bidding suppliers have spent considerable resources and cost, which could have been avoided by more communication about impact of the availability requirement throughout the tender process.

3.2.3 Case #3: the danger of ignoring unspecified NFRs

A very large project is underway to deliver a transaction processing system for a property services company. Within the contract, the specification did not include any requirements for performance, availability or any other NFR. In hindsight, the specification did not accurately define the quantity of information to be processed or accurately express the complexity of the business process the client actually needed. The project begins work and some members of the team begin to voice concerns amongst themselves about the growing complexity of the functional requirements and growing data requirements, and in particular the potential for this combination of factors to adversely

impact transactional performance experienced when the functions are implemented, fearing that the customer will find the solution to be unacceptable. These concerns are not raised with the customer. The team attempts to gain acceptance of the solution on functional grounds only, strictly fulfilling the contractually agreed specifications.

At the stage of formal acceptance testing, the customer states that he is very unhappy with the performance of the application and says that it will not viably satisfy his needs. The project team points out that no contractual obligation exists to require the supplier to deliver in-line any particular non-functional requirements. The customer steadfastly refuses to accept the system and stalemate is reached - a situation that could have been avoided by recognizing and communicating about the implicit performance requirement at an early stage.

3.2.4 Dilemmas for suppliers

In our experience, there are two flavors of dealing with NFRs in RfPs: they are either not mentioned at all (Cases #1 and #3), or they are present as hard, quantified requirements (Case #2), often poorly and ambiguously stated. Both flavors lead to dilemmas when writing a proposal in response to the RfP.

If an RfP contains hard-quantified quality attribute requirements for systems that are newly to be designed, the dilemma is caused by the uncertainty in the cost of fulfilling them. The level of uncertainty is often much larger than many stakeholders realize. New architectural combinations may have to be tried out with highly unpredictable effects on a number of interacting quality attributes. In a tendering situation, there often is no time to reduce the uncertainty by executing e.g. a proof of concept. This leaves the supplier with essentially two options: either going along with the requirement and taking on the full risk of the uncertainty, or offering a non-compliant solution - thereby risking losing the job. This is essentially a regular risk management issue - except that, as stated before, the risk in NFRs is that they can very considerably stress projects.

For quality attributes that are ignored in RfPs, the dilemma to the supplier is of a different kind. Their professionalism leads them to take into account these attributes even if no quantified NFRs are present, but how far should they go with this? Too much attention may inflate the price, potentially causing the bid to be lost due to quality attributes that are not even formally required by the customer. Too little may lead to severe problems later on, because customers have expectations about quality attributes, even if they are not explicitly quantified in the RfP (see Case #3). There are well-documented court cases [RACV Insurance Pty Ltd v. Unisys Australia Ltd, 2001] that show that suppliers have a duty of care in this area that can go beyond the contractually explicit requirements.

NFRs, whether documented in the RfP or not, are a regular source of dilemmas for

suppliers responding to RfPs. One sometimes gets the impression that the tendering rules force customers to contract the supplier that has the lowest level of understanding of the NFRs. A supplier that is insufficiently aware of the impact of NFRs will generally submit a lower priced offer, because other suppliers will calculate the proper cost of addressing the NFRs, or the proper contingency needed to deal with NFR-related risks. Of course, by the same token, the NFR-unaware supplier that wins the bid will subsequently perform poorly in terms of quality attributes, and probably overrun delivery time and budget once the NFR trouble has come to light. This impression is confirmed by the example of the Dutch highway tunnel safety systems [Gram and Keulen, 2010]. The project was plagued by quality issues so severe that they caused years of delay. The government committee that investigated the trouble reports that "at the time of awarding the bid, it was known that the winning bidder scored quite badly on quality [...], but the quality criterium weighed insufficiently to compensate for the low price. The winning party, when asked, confirmed that, in their opinion, they could realize the project." [Gram and Keulen, 2010]. Due to European tendering rules [European Commission, 2004], in this situation the customer would not even have been *allowed* to award the bid to another supplier. This is a clear example of a supplier that won a bid due to a lack of NFR-awareness combined with tendering rules.

3.3 NFR Quantification as an Economic Problem

NFRs in RfPs can be expressed to various degrees of (un)certainty. They can be documented as vague goals that still need to be clarified and disambiguated, like the softgoals of [Chung et al., 1999]. They can also be expressed in quantified values. A lot of literature is available on the benefits of quantifying NFRs. Crisply quantified NFRs give architects a basis for their design decisions [Bass et al., 2003, Gilb, 2005], allow architectures to be validated [Clements et al., 2002], and give testers and customers a firm basis for acceptance testing [Pinkster et al., 2004]. There is no dispute that the most important quality attributes for a system should, at some point in time, be quantified in terms of objectives, targets and eventually (acceptance) test criteria. The ISO-25000 standard [ISO/IEC 25000, 2005] provides a model and metrics to do so, and several approaches exist [Barbacci et al., 2003, Gilb, 2005] as alternatives or supporting processes for these standards.

From an economic perspective, NFR quantification can be seen as an exercise in optimizing the value/cost ratio. Quantified NFRs have to be related to two economic entities: the business value of the realized NFRs (quality attribute) to the customer, and the cost to the supplier to realize the NFR (which in turn translates to price to the customer). These relationships have been extensively explored by [Kazman et al.,

2002] and later by [Regnell et al., 2008] and [Berntsson Svensson, 2009], and will be explained briefly below.

3.3.1 Cost

Quality requirements tend to be very cost sensitive. This is because NFRs are fulfilled through architectural strategies and choices, such as technology selection or layering styles [Bass et al., 2003], which usually affect more than one Quality Attribute. These architectural decisions are usually discrete choices between alternatives, each carrying their own cost. These discrete choices cause discontinuous jumps in the relationship between quality attributes and cost [Regnell et al., 2008], as illustrated in the "Cost vs. Max Response Time" graph in Fig. 3.1(a). This relationship is called the "cost function", and it is determined by the architectural decisions influencing the NFR. At the time of writing the Request for Proposal, the cost function is actually unknown by the customer because the architectural choices have not been explored in depth (this is the job of the supplier). At bidding time, even the supplier usually does not have the time to sufficiently explore the cost and time needed to fulfill NFRs. For newly designed systems, figuring out the true cost function of NFRs often requires extensive model calculations or architectural prototyping, for which the deadline of tender submission is usually far too short. From the supplier's point of view, the affordability factor is acutely felt at bid stage, where the matter of competitive positioning is uppermost in their mind.

3.3.2 Value

The business value of quality is often a highly intractable entity [Garvin, 1984], which is illustrated in Case #1 above. Who can calculate the difference in value between a system with 99.99% availability and one with 99.999% availability? The difference in cost between the two may be prohibitive, as is illustrated in Case #2. The relationship between quality attribute and value is called the "value function". The value function is typically stakeholder-dependent: e.g. improving performance by 50% at night-time may be worth a lot to a particular department, while another department in the same company derives more value from increased security resilience or maintainability. In Fig. 3.1(b), an example of a relationship between business value and maximum response time for a function is depicted. At bid time, this relationship is unknown to the supplier: even if the response time requirement is quantified in the RfP, it is a single number or a statistical spread of numbers, but rarely explicitly related to the business value.

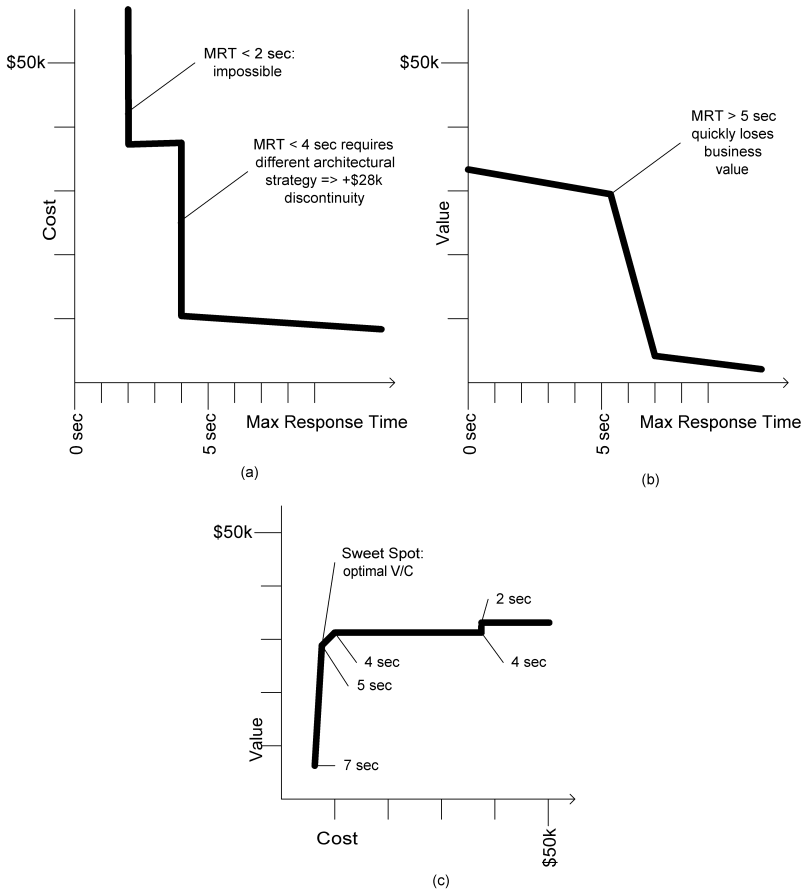


Figure 3.1: Balancing cost and value

3.3.3 Balancing cost and value

In Fig. 3.1(c), a Cost versus Value graph is derived from two underlying relationships for a particular quality attribute: the Maximum Response Time (MRT) for a function. Assuming that we want to maximize the Value/Cost ratio, finding the "sweet spot" in this graph is easy: the point on the graph that has the steepest straight line to the origin represents the optimal quantified NFR from an economic point of view. We have to keep in mind, though, that the optimization we have performed here concerns only one NFR. In reality, quality attributes are not orthogonal, so a full cost/benefit analysis would require a more complex, multidimensional calculation involving cost and value functions of all relevant quality attributes.

We now have a simplified method to quantify NFRs from an economic perspective. Three things are essential to this method:

- Supplier knowledge of the NFR's cost function.
- Customer knowledge of the NFR's value function.
- Communication of said knowledge between customer and supplier.

Even though these essentials were derived from a simplified method, it is easy to see that they are needed for any realistic approach to quantifying NFRs in a way that makes economic sense.

3.4 NFR Quantification as a Negotiation Problem

All three of the essentials mentioned in the previous section are usually low at tendering time, and significantly increase only after the contract has been signed:

Cost function knowledge increases by the research and experience of the supplier's delivery team.

Value function knowledge increases as end-users and business managers of the customer organization get more involved in the execution of the project and see more and more of the solution at work.

Communication between customer and supplier is severely restricted at tendering time, and gradually opens up after contract signing, as mutual trust grows.

So the reasonable thing to do is to postpone the quantification of NFRs until after the contract signing, when there is a relationship between customer and supplier that

allows free exchange of information, and sufficient time to elaborate architectural alternatives and establish their costs. However, uncertainty in NFRs implies significant risk. It is natural for customers to seek as much certainty as possible that the system's quality attributes will fulfill their needs. The natural tendency therefore is to demand a supplier's commitment to fixed and quantified NFRs. This puts the supplier in a difficult position: should they refuse to commit, or convert the inherent risk into a contingency premium on top of the price? Either way may lead to not winning the bid, due to either non-compliance or overpricing.

The risk and cost of NFRs often become objects of contract negotiations. This does not help the three essentials mentioned above, as customer and supplier now have to deal with negotiation tactics such as risk avoidance, divide and conquer, good guy/bad guy, salami nibbling and slicing, on top of the technical difficulties of the engineering process. Especially communication of cost and value aspects between customer and supplier falls victim to the commercial necessity of playing ones cards close to the chest.

We thus come to the core of the issue: from an engineering and economic perspective, NFRs should not be quantified until cost and value knowledge and customer/supplier communication have been sufficiently established, which usually occurs well after contract signing; on the other hand, commercial reality often demands quantified NFRs committed to in the contract. In the next section, we will explore some possible solutions to this issue.

3.5 Towards Solutions

In this section, we will present two approaches that can help alleviate the issues around NFR quantification in a commercial setting: Requirements Convergence Planning and Competitive Dialogue.

3.5.1 Requirements convergence planning

As stated in §3.2.4, when responding to an RfP containing hard-quantified NFRs, suppliers with insufficient assurance that the requirements can be met basically have two options:

Scenario A Respond "compliant" and deal with the resulting risk.

Scenario B Respond "non-compliant", and offer an alternative for addressing the underlying stakeholder needs.

We asked the Logica Architecture Community of Practice for their views on these scenarios in an open e-mail question, and received a dozen responses. The anecdotal evidence in these responses led to the following:

> In *Scenario A*, the risk is usually dealt with by increasing the contingency budget, and mitigated by adding assumptions about the interpretation and measurements of the quantified NFR. This leads to a higher price for the customer, and a remaining chance that the supplier cannot achieve the required number (usually performance or availability). Failure sometimes means penalties, sometimes budget and delivery time overruns. As one respondent writes, "we usually get away with it".
>
> Most respondents prefer *Scenario B*, but indicate that only "mature" customers will agree to it - customers who are aware of the intricacies of NFR quantification. Scenario B requires room for discussion in which the supplier can highlight to the customer that particular requirements can be very expensive, and in which the hard requirement can be moved to a "target value". Instead of committing to the NFR value quantified by the customer, the supplier will commit to a *process* to find a proper balance of affordability, i.e. a number that is acceptable to the customer and achievable at reasonable cost. This process is sometimes called "calibration" or "clarification".
>
> The choice between Scenarios A and B is usually based on the following aspects:
>
> - Possibility to respond "non-compliant" without automatic disqualification.
> - Contractual conditions in case of not making the NFR (e.g. penalties).
> - Customer's openness, awareness of NFR criticality and (assessed) willingness to compromise on the NFR if the project runs into trouble.

In recent years, we have started to call the process referred to in Scenario B *Requirements Convergence Planning*. A Requirements Convergence Plan (RCP) is a plan to quantify specific quality attributes that cannot be committed to at contract signing time. This plan sets out a process of discovery and ultimately convergence on quantification of performance or other NFRs with open collaboration from a customer. The plan seeks to identify the most favorable balance of value and cost for performance attributes, whilst implicitly reducing risk.

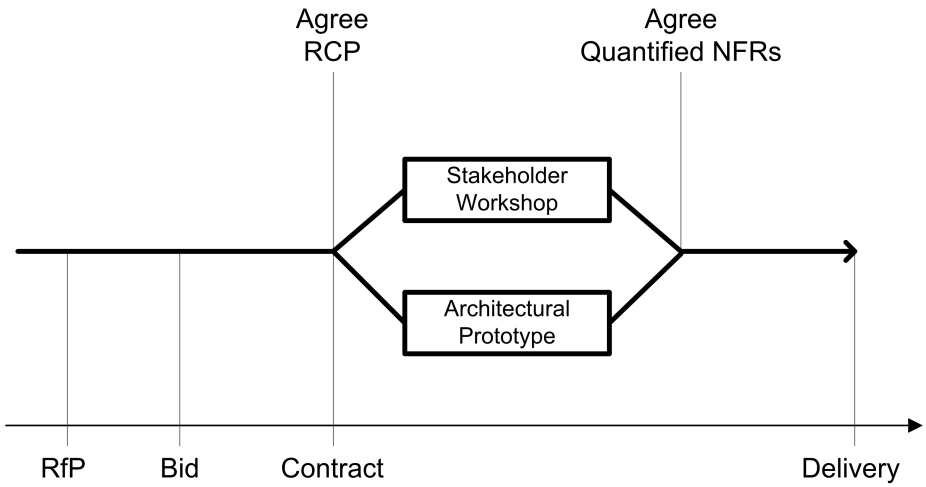


Figure 3.2: Requirements Convergence Plan

One way of planning requirements convergence is by defining a process of incrementally benchmarking architectural strategies and sharing the results of the benchmarks with the customer. In this way the customer becomes fully aware of what can feasibly be achieved within the cost constraints of the project, what risks this entails, and the impact of delivery timescales of striving for a different balance.

The RCP concept is visualized in an example in Fig. 3.2. At contract signing time, customer and supplier agree to the execution of the RCP, which in this case contains two activities: a series of stakeholder workshops (e.g. Quality Attribute Workshops [Barbacci et al., 2003]) to increase knowledge about the business value of the quality attributes, and an architectural prototype to research what quality attribute level can be achieved at what cost. Both customer and supplier are involved in both activities, stimulating the flow of information needed to make the trade-offs. At the end of the RCP period, the results of the stakeholder workshops and prototype evaluation are put together, and result in a firm quantified NFR. The supplier then commits to delivering the solution fulfilling the NFR.

Requirements convergence planning can be called a *two-stage commitment* approach for NFR quantification: at contract signing, the supplier does not commit to a quantified NFR, but to the execution of the RCP. At the end of the RCP, once a mutually agreed balanced NFR is achieved, it is signed off and committed to. The details of what happens in the RCP can be worked out on a case-by-case basis, as long as

the three essentials mentioned in §3.4 above are sufficiently addressed: cost function knowledge, value function knowledge and communication between customer and supplier about them.

Apart from getting close to economically optimized NFR quantification, a benefit of this approach is that it allows the supplier to more keenly price its offer, as the performance evaluation exercise is openly effort-boxed and no explicit commitment is made to meeting specific NFRs at time of tender. The additional advantage to the customer is that he is not paying for the possibly large contingency that a supplier would otherwise have to load his offer with if this process were not to be followed.

This approach can not always be applied, since it requires an RfP that allows it to be proposed. Also, the customer must be willing to give up the certainty of a committed and quantified NFR, in exchange for the probability of better value for money spent on achieving NFRs. We have had some success with customers that welcome the openness, feel that they are more likely to get what they need and feel that they will not necessarily pay overly for it. Suppliers feel better in control of the risks, and feel as though they are in a better position to satisfy the customer's needs and make a profit.

3.5.2 Competitive dialogue

In 2004, the European Council added a new tendering procedure for the public sector, called "Competitive dialogue": *a procedure in which any economic operator may request to participate and whereby the contracting authority conducts a dialogue with the candidates admitted to that procedure, with the aim of developing one or more suitable alternatives capable of meeting its requirements, and on the basis of which the candidates chosen are invited to tender*. [European Commission, 2004] The Competitive Dialogue is meant for "particularly complex contracts". The aim of the dialogue is to "identify and define the means best suited to satisfying their needs. They may discuss all aspects of the contract with the chosen candidates during this dialogue." The Competitive Dialogue tendering procedure contains significantly less restrictions in the communication between customer and supplier at tendering stage. [US Government, 2005] contains a form of tendering called "Contracting by Negotiation", which was introduced in the regulations in 1997; like Competitive Dialogue, it has less communication restrictions than its counterpart, Sealed Bidding.

The fact that Competitive Dialogue allows a freer exchange of information between customer and suppliers makes it more suitable than the previously existing procedures for an integrated RE/AD approach in the IT solution domain. In practice, we see more and more use of the competitive dialogue tendering procedure, but it is still only applied in a minority of tenders: in 2011, only about 5% of the IT tenders Logica is interested in follows the competitive dialogue procedure. The following anonymous example shows

how such a dialogue is typically conducted:

> A government ministry is looking to outsource the operation and maintenance of its Enterprise Resource Planning (ERP) application landscape to an IT service provider. The contract will be for 5 years, with a total contract value in the order of magnitude of 50M€. A Request for Information (RfI) goes out, after which a shortlist of suppliers is selected. An RfP based on the competitive dialogue model follows 5 months after the RfI. Candidate suppliers have two months to register for the bid. A six month competitive dialogue phase then starts. During the competitive dialogue phase, there are 8 workshops with each supplier. The objective of the workshops consists of two business goals: cost reduction and flexibility enhancement. Suppliers are asked to use the workshops to bring forward ideas so that the ERP landscape can be operated and maintained in a less costly and more flexible manner. Even though the original RfP contains quite detailed requirements (including many NFRs), suppliers are actively encouraged to think outside of the boundaries of these requirements during the competitive dialogue phase. The requirements may be adjusted as a result of the workshop outcome, in order to obtain the business goals. Bidders taking part in the competitive dialogue get part of their costs reimbursed.

In this example, we see that the stakeholder workshop part of the requirements convergence plan (§3.5.1) is in place. In similar cases, suppliers are also asked to provide a proof of concept - analogous to the architectural prototype in the requirements convergence plan. In other words, the four central ideas that make up requirements convergence planning are used in practice in tendering situations: two-stage commitment, stakeholder workshops, architectural prototyping and cusomter/supplier dialogue.

3.6 Discussion and Conclusions

In this chapter, we have presented some key issues related to NFR quantification in customer/supplier relationships. Critical NFRs should be quantified, but we should *beware of premature quantification*: as our real-life examples illustrate, prematurely quantified NFRs can cripple projects and lead to diverging points of view in customer/supplier relationships that are very hard to resolve.

We have concluded that, in most cases, it is impossible to find the optimal (best value/cost ratio) quantification for important NFRs at tender time. Optimal quantification requires sharing of information between customer and supplier, and it requires

time to establish at least a reasonably proven estimate for the cost and value relationships. One possible way to create better NFR quantification circumstances for customers and suppliers is by means of a requirements convergence plan, which we will encounter again in Chapter 9 as a practice in our solution architecting approach RCDA. The European Union has a new tendering procedure that can be used for requirements convergence, "Competitive Dialogue".

With the ever growing complexity of IT systems and projects, predicting system quality attributes becomes increasingly harder. Academia and industry are researching ways to improve this predictability, but they cannot win this race while the complexity of IT systems and projects increases at its current frantic rate. In the mean time, we have to deal with an imperfect world. There is no unambiguous recipe for balancing cost and value of quality attributes. Performing the balancing act while negotiating a contract is fraught with uncertainty and danger, and can even lead to failure of IT projects. The industry could benefit from a change in attitude that reflects this state of affairs. Transparency and awareness between customers and suppliers about NFRs is one part of that attitude; willingness to share the risk of unquantified NFRs is another. Both transparency and risk sharing require a basis of trust to exist between customers and suppliers in the IT industry. Without this trust, formal requirements documents or contracts with precisely quantified NFRs will not help to guarantee success.

3.6.1 Related work

Negotiating and risk balancing

Viewing NFRs as a negotiation problem was first introduced in the WinWin Spiral model of Barry Boehm et al [Boehm et al., 1995, Boehm and In, 1996]. The WinWin Spiral model is an iterative process of negotiating requirements between stakeholders, based on win-conditions. More recently, [Fricker et al., 2010] introduces the use of Implementation Proposals to facilitate the negotiation and understanding between stakeholders and architects.

The need for iterating between stakeholders to resolve requirements conflicts and reach agreement is also described in the elaboration phase of the Unified Process [Kruchten, 1998]. The issues that such elaboration iterations raise in relationship to a tendering situation have been recognized by others, and Pitette proposes a solution involving Progressive Acquisition [Pitette, 2001]. Another discussion of the difficulties of requirements specification in RfPs can be found in [Paech et al.], which reports on the experiences of a supplier in a tender process, identifies challenges and presents some possible solutions for the supplier.

An extensive treatment of balancing the forces of risk and timing can be found

in [Karolak, 1995].

(Un)certainty in requirements

For dealing with uncertainty in requirements, two approaches appear in literature: modeling the uncertainty [Laplante and Neill, 2005, Noppen, 2007], and tuning the development process to better deal with change, which is one of the basic premises of the Agile movement [Agile Alliance, 2001]. The gradual increase of certainty during IT projects (§3.4) is often visualized as a "cone of uncertainty" [Mcconnell, 1997]. It was first described in [Boehm, 1981] as the "funnel curve".

[Davis, 2005] gives lots of practical advice on how to prevent overspecification of requirements. [Glinz, 2008] presents another economic perspective on NFR quantification, focusing on the risk-based need to quantify versus the cost of the quantification activities.

NFR trade-off approaches

We have seen in §3.3 that economic reasoning about NFR quantification requires knowledge of the various architectural strategies that influence the NFRs. In other words, we see that economic justification of NFR quantification requires knowledge of the solution architecture. This confirms the need for an integrated approach for requirements engineering and solution architecture as identified previously by [Paech et al., 2002]. As stated before, the one-dimensional value/cost trade-off method presented in §3.3 is a simplification: we use it here because it allows simple reasoning about quantifying an NFR, and clearly demonstrates the need for the "three essentials" for proper quantification: cost knowledge, value knowledge and communication. Almost the same method is used in QUPER [Regnell et al., 2008], who apply it in the context of product roadmapping. Several approaches for *multi*-dimensional trade-off also exist, such as CBAM [Kazman et al., 2002] and the NFR Framework [Chung et al., 1999, Lamsweerde, 2009]. [Supakkul et al., 2010] classifies such approaches as "selection patterns" and compares a number of them. All of these are more sophisticated than the method presented in §3.3, but in the end all require the same "three essentials".

[Fricker and Glinz, 2010] presents a case study analyzing and monitoring the handover and negotiation process between stakeholder and architect. The case study reports that a substantial part of the requirements change after the solution is presented to the stakeholder: the intended solution changed the stakeholder's position and triggered substantial requirements modifications "to exploit strengths and account for weaknesses of a possible solution". They report that "requirements understanding was perceived good-enough only after negotiation". Even though that study was not specifi-

cally targeted at NFRs, the results strongly confirm our position that proper NFR determination requires knowledge of the solution architecture.

[Gilb, 2005] appears to be a strong opponent to this position, advocating *'How Good' and 'How Much' before 'How'* as a matter of principle: "All performance requirements and resource requirements must be stated before any design idea can be fully and properly evaluated." However, in the same list of principles Gilb also states: "You cannot have correct knowledge of all the interesting requirements levels for a large and complex system in advance," indicating at least partial agreement with our position. The Design Engineering process presented in [Gilb, 2005] requires the same "three essentials" of cost knowledge, value knowledge and communication to work, indicating that it too would suffer from the communication limitations often encountered in formal client/supplier relationships.

Summarizing, we have found no existing method in industry and literature that allows proper quantification of NFRs in a situation with severe communication constraints between customer and supplier. The only way to address the issues highlighted in §3.2 is to use contract negotiation models with less constraints, taking into account the characteristics of NFR quantification.

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. IT projects where modifiability is perceived to have low business criticality lead to consistently high customer satisfaction. Our conclusion is that modifiability deserves more attention than it is getting now, especially because in general it is quantified and verified considerably less than other NFRs. Furthermore, IT projects that applied NFR verification techniques relatively early in development were more successful on average than IT projects that did not apply verification techniques (or applied it relatively late in development).

4.1 Introduction

As we saw in Chapter 1, NFRs represent a promising area for improvement, because dealing with NFRs is viewed as a particularly difficult part of requirements engineering [Berntsson Svensson, 2009], and 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].

One could say that architects are responsible for facilitating and realizing NFRs during software development; they are the population that has to "deal" with NFRs. Knowledge about how architects perceive and address NFRs can help IT organizations improve their architecting practices and project success rates. Therefore, we set up a

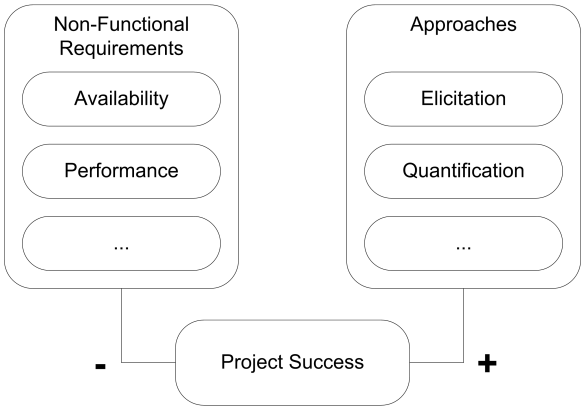


Figure 4.1: Conceptual model

survey among the members of Logica's architecture community of practice to gather such knowledge. The survey was aimed at investigating how architects perceive the importance of NFRs, and which approaches they use to deal with them. We were also interested to see whether we could link these findings with IT project success.

4.1.1 Conceptual model

The context of this study is IT Development Projects, defined as *a project where an IT system (application, software, infrastructure or other IT system) is designed, constructed and implemented.*

The focus of the survey is on investigating the two relationships depicted in the conceptual model, shown in Fig. 4.1, within the context of bespoke solution development, and from the perspective of the architects. On the one hand, the more important non-functional requirements are, the greater the implied risk to IT project success if they are not fulfilled. On the other hand, several NFR approaches could help an IT project deal with NFRs. To put it another way, the assumption is that IT project success depends on the importance of the NFRs and the application of approaches for dealing with NFRs. We are interested in the following questions, which are an elaboration of RQ-1c:

1. How do architects perceive the importance of non-functional requirements?

2. Is there a significant relationship between the perceived importance of non-functional requirements and IT project success?

3. What approaches for dealing with non-functional requirements do practitioners apply?

4. Is there a significant relationship between applying approaches for dealing with non-functional requirements and IT project success?

A complicating factor in this model is the fact that we are by necessity looking at all this through the architect's eyes. Since the measuring instrument is a survey among architects, we are not actually measuring the importance of NFRs, but rather the *architect's awareness* of their importance. Architecture is a risk driven discipline [Fairbanks, 2010]. Awareness of a risk is a prerequisite to dealing with it. The more an architect is aware of the importance of a requirement and its implicit risk of not being fulfilled, the better he is able to address it. This mechanism works against the expected negative impact of NFR importance on project success; it can even completely negate the negative impact when the architect is fully successful in addressing the NFRs he is aware of.

4.2 Survey Description

The core of this study is an on-line survey that was conducted in 2010 among practicing architects. In addition to the survey itself, we organized two expert workshops, consisting of a guided discussion with a select group of architecture experts in Logica NL. One workshop was held prior to the survey itself, and its prime objective was to align the survey's contents with the vocabulary and way of working within Logica. The second workshop was held after the survey, and its purpose was to enrich the initial quantitative analysis results with qualitative knowledge from practicing architects.

The invitation to participate in the survey was sent out by e-mail to around 350 members of the Netherlands (NL) Architecture Community of Practice (ACoP) of Logica. The ACoP consists of experienced professionals practicing architecture at various levels (business, enterprise, IT, software, and systems architecture) in project or consultancy assignments. The survey was closed after 16 days. By that time, 133 responses were collected. After elimination of duplicates (1), incomplete responses (51) and responses from respondents that indicated they had not fulfilled the role of architect on their latest project (41), 39 responses remained.

The survey consists of 23 questions divided over four sections. The first section consists of questions that are related to the general characteristics of the latest completed project of the respondent. The second section asks the respondent to evaluate the success of his or her latest completed project from a number of perspectives. Respondents were asked to characterize their latest completed project in terms of NFRs in the third section of the survey. The fourth section evaluates the approaches deployed

for managing and dealing with NFRs in their latest completed project. The survey concludes by presenting a number of statements about NFRs to the respondent. Examples of what the survey questions looked like are shown in Fig. 4.2.

4.2.1 Constructs

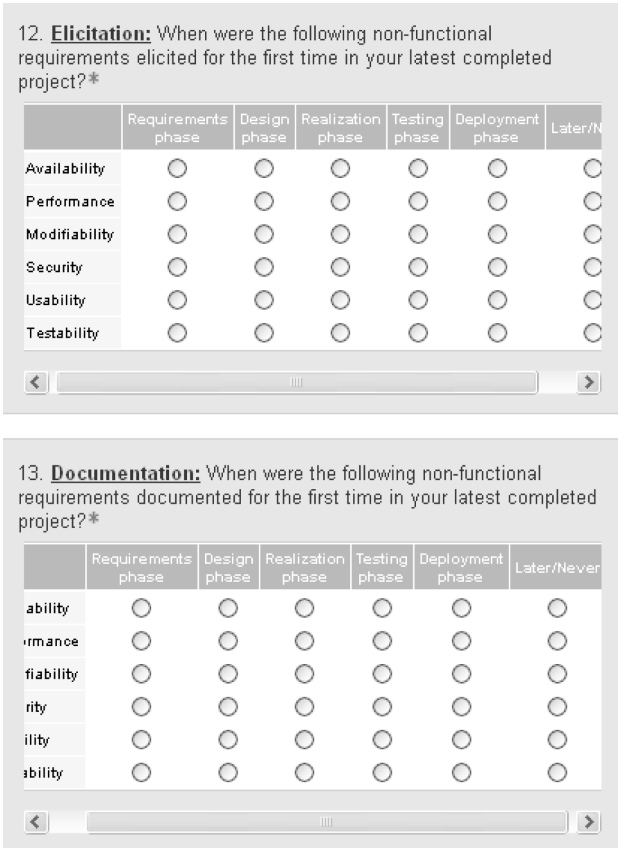
Considerable time and effort was spent on translating the key concepts of the conceptual model into operationalized constructs for use in the survey. The four key concepts were *Non-Functional Requirements*, *NFR importance*, *project success* and *NFR approach*. Each of these concepts was first operationalized by looking for useful descriptions and classifications in literature, which resulted in a draft survey. The draft survey was then the subject of an expert workshop, in which it was discussed by eight architecture experts from Logica's NL central technical unit (a kind of architecture board). The constructs were the main topic of the workshop discussion - especially the use of terms and models that would be commonly understood by the company's architecture community. The workshop outcome led to a modified, final version of the survey.

Non-Functional Requirements

The Non-Functional Requirements concept had to be made more specific. To be able to analyze the impact of different NFRs, the NFR concept had to be classified into subtypes. The problem of choosing a specific scheme to sub-classify NFRs lies in the observation that even well-known classification schemes are terminologically and categorically inconsistent with each other [Chung et al., 1999, Mairiza et al., 2010]. Many of the published classifications and definitions of NFRs have their own communities in science and practice [Bass et al., 2003]. Since a significant number of architects in Logica had been trained in the software architecture practices of the Software Engineering Institute, the six most common and important types of NFRs distinguished by those practices were used in the survey. They all refer to quality attributes. Their basic descriptions were taken from [Bass et al., 2003], and were slightly enhanced with examples by the pre-survey expert workshop to increase understandability in the architecture community context:

Availability concerns system failure and its associated consequences. A system failure occurs when the system no longer delivers a service consistent with its specification. Such a failure is observable by the system's users (either humans or other systems). Reliability and recoverability are examples that belong to this type.

Performance events (interrupts, messages, requests from users, or the passage of time) occur, and the system must respond to them. Performance is concerned



12. **Elicitation:** When were the following non-functional requirements elicited for the first time in your latest completed project?*

	Requirements phase	Design phase	Realization phase	Testing phase	Deployment phase	Later/N
Availability	○	○	○	○	○	○
Performance	○	○	○	○	○	○
Modifiability	○	○	○	○	○	○
Security	○	○	○	○	○	○
Usability	○	○	○	○	○	○
Testability	○	○	○	○	○	○

13. **Documentation:** When were the following non-functional requirements documented for the first time in your latest completed project?*

	Requirements phase	Design phase	Realization phase	Testing phase	Deployment phase	Later/Never
ability	○	○	○	○	○	○
rmance	○	○	○	○	○	○
fiability	○	○	○	○	○	○
rity	○	○	○	○	○	○
ility	○	○	○	○	○	○
ability	○	○	○	○	○	○

Figure 4.2: Example survey questions

with how long it takes the system to respond when an event occurs. Efficiency and throughput are examples that belong to performance.

Modifiability considers how the system can accommodate anticipated and unanticipated changes and is largely a measure of how changes can be made locally, with little ripple effect on the system at large. Adaptability, maintainability and compatibility are examples that belong to this type.

Security is a measure of the system's ability to resist unauthorized usage while still providing its services to legitimate users. An attempt to breach security is called an attack and can take a number of forms. It may be an unauthorized attempt to access data or services or to modify data, or it may be intended to deny services to legitimate users.

Usability is concerned with how easy it is for the user to accomplish a desired task and the kind of user support the system provides. It can be broken down into the following areas: learning system features, using a system efficiently, minimizing the impact of errors, adapting the system to user needs, increasing confidence and satisfaction.

Testability refers to the ease with which software can be made to demonstrate its faults through (typically execution-based) testing.

NFR importance

How does one measure the importance of each type of NFR for a project? The experts in the pre-survey workshop agreed that simply asking for the number of requirements for each type of NFR is not valid. Intuitively, a project could have only a few performance requirements that are nevertheless critical for the system. Conversely, it could have more requirements of another type that are not critical. Furthermore, when you measure the number of requirements for each type of NFR, you are only measuring NFRs that were documented or elicited. The problem with NFRs often is that certain NFRs are *not* documented or elicited. Therefore, the suggestion of the experts was to use the concept of *business criticality*: a certain type of NFR is more important if it is relatively more critical for the system and the business of the customer. This is a concept that can be judged by the respondent in hindsight and is more valid than a simple requirement count. An NFR is considered business critical when it is vital to the customer's business. The measure in which highly business critical NFRs are fulfilled has a high impact on the system's business value, and vice versa. Respondents were asked to rate the business criticality of each of the six types of NFRs on a 5-point Likert-scale (very low, low, medium, high, very high).

Project success

Project success has long been an active research topic. Traditionally, project success is defined in terms of meeting time, cost and quality objectives [Pinto and Slevin, 1988]. More recently, it has been observed that projects can be successful in ways that cannot be measured by these traditional criteria. Based on these insights, [Baccarini, 1999] have constructed a conceptual framework for project success. Baccarini's framework distinguishes between *Project Management Success*, which includes the three traditional criteria of time, cost and process quality, and *Product Success*, which adds criteria related to the product in a more strategic way, involving the product's goal and purpose and product satisfaction. Team Satisfaction in Baccarini's framework can relate to both project and product; in our experience, this is especially true for architects, who derive a large part of their job satisfaction from product quality. This observation is confirmed by research by Linberg et al. [Linberg, 1999] and more recently by Procaccino et al. [Procaccino, 2005], who observe that developers' perception of project success often deviates significantly from the traditional criteria. Developers (including architects) tend to judge success by criteria that extend beyond the project, sometimes to the extent that even canceled projects can be successful in their eyes.

Our project success construct consists of five dimensions, that are designed to reflect the interests of the three main stakeholders (cf. [Dvir et al., 2003]). Meeting time and budget corresponds to project success from a managerial perspective, as does efficient use of resources. Customer satisfaction is included to reflect the perspective of the customers, and solution quality is the dimension that measures the success from the perspective of the development team. Respondents are asked to rate the success of their latest completed project in terms of these dimensions on a 5-point Likert-scale (very unsuccessful, unsuccessful, neutral, successful, very successful). The overall project success parameter is the sum of the responses for the 5 values. Cronbach's α [Cronbach, 1951] was used as a reliability test to assess internal consistency of this construct; at $\alpha = .858$, the construct proves to be valid ($> .8$).

NFR approach

The survey asks the respondents to indicate what approaches were applied for dealing with NFRs during their latest completed IT project. Practitioners find dealing with NFRs the most difficult part of requirements engineering [Berntsson Svensson, 2009]. The need for ways to manage NFRs has led several researchers to propose methods and techniques for dealing with NFRs. A set of similar methods and techniques, related to the same requirements engineering activity, that can be used to deal with or manage NFRs (or requirements in general) is defined as an *NFR approach*.

[Berntsson Svensson, 2009] and [Paech and Kerkow, 2004] both provide classifications of activities aimed at dealing with NFRs. After merging these two classifications and discussing the result in the pre-survey expert workshop, the following approaches were included in the survey:

Elicitation interacting with stakeholders (customers, users) of a system to discover, reveal, articulate, and understand their requirements.

Documentation requirements are written down in order to communicate them to stakeholders (designers, developers, testers, customers).

Quantification NFRs are made explicit by giving them numbers on a measurable scale. This makes the NFRs verifiable.

Prioritization assigning priorities among the different NFRs on the basis of their relative importance.

Conflict analysis identifying the interdependencies and conflicts among the NFRs.

Verification verifying that a system fulfills requirements, e.g. by prototyping, simulation, analysis, testing or other means.

For a full operationalization of the NFR Approach construct, we not only need a classification of sub-types, but also a way to measure their usage in the projects. The simplest way to determine which of the approaches were applied would be to ask respondents using a yes/no format. However, this is not sufficient. We want to be able to distinguish between situations where the approaches were used early on in the project ("on time") and late in the project ("after the fact"). Several studies [Westland, 2002, Grady, 1999] have pointed out that the relative costs of correcting (requirements) errors increases during the development life cycle. In line with these findings, one may expect that applying an approach later in the development life cycle is less effective; in other words, the earlier an approach for dealing with NFRs is applied, the stronger its positive impact on project success is expected to be. Therefore, respondents are asked to indicate *when* the approaches were applied during the development life cycle for each type of NFR on a 6-point Likert-scale. The Likert-scale represents five phases of a generic systems development life cycle (requirements phase, design phase, realization phase, testing phase, deployment phase) and a later/never option.

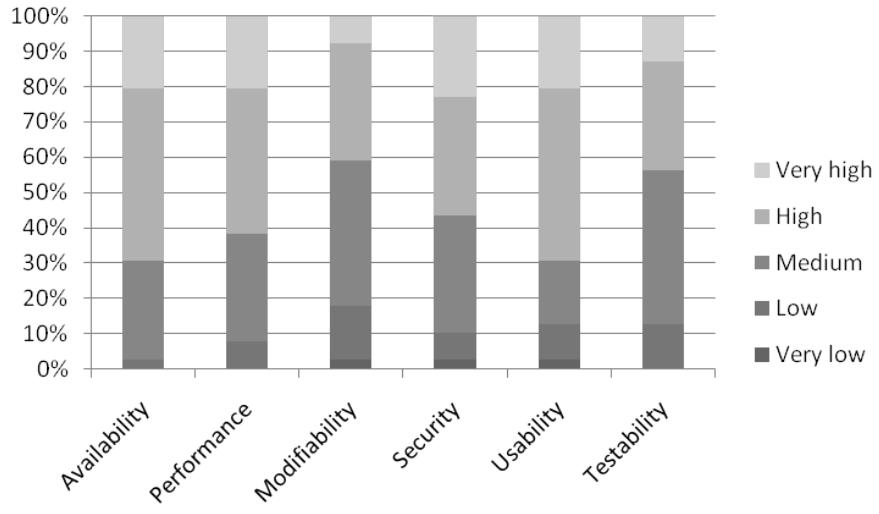


Figure 4.3: Perceived business criticality of NFRs

4.3 Analysis of Survey Responses

In this section, we present the most interesting results of the quantitative analysis of the survey responses. The outcome of this quantitative analysis was discussed by a post-survey workshop with architecture experts in Logica NL. The results of this post-survey workshop will be presented in the Discussion section of this chapter.

In Fig. 4.3, an overview is given of how the software architects rated the business criticalities of the NFRs.

Availability and (to a slightly lesser degree) usability are generally considered highly business critical, while modifiability and testability score relatively low. Performance and security are somewhere in the middle.

Overall, the types of NFRs are almost never unimportant: very few respondents rated the business criticality of any type of NFR as very low or low. This suggests that each type of NFR has at least some basic level of business criticality in every project. Therefore, each project involves dealing with every type of NFR at least to some degree.

Table 4.1 shows how many of the 39 architects applied each of the approaches, dif-

Table 4.1: Application of approaches per NFR

	availability	performance	modifiability	security	usability	testability	Total
elicitation	37	37	32	37	35	34	212
documentation	32	30	25	32	25	26	170
quantification	32	29	16	30	20	21	148
prioritization	26	27	18	24	19	17	131
conflict analysis	22	24	15	20	18	16	115
verification	29	33	18	33	28	30	171
TOTAL	**178**	**180**	**124**	**176**	**145**	**144**	

Table 4.2: NFRs, correlation coefficient with IT project success

Type of NFR	**Kendall's τ**	**Sig. (1-tailed)**
Availability	.086	NS
Performance	-.181	NS
Modifiability	-.257	.023
Security	.078	NS
Usability	-.102	NS
Testability	.095	NS

ferentiated per NFR. Again, modifiability scores low: almost all approaches are applied less for modification than for other NFRs, especially quantification and verification.

4.3.1 Non-functional requirements and project success

Based on the theory described earlier, the expectation is that the business criticality of NFRs is negatively correlated with IT project success, but that this effect may be dampened by the architect's awareness bias. For each NFR category, this hypothesis is tested using Kendall's τ (one-tailed) and the level of statistical significance is .05 (α = .05). The value of Kendall's τ ranges between -1 (perfect negative correlation) and +1 (perfect positive correlation).

A summary of the results is presented in Table 4.2. Statistically, we should ignore correlation coefficients where the significance $Sig. > .05$, which are indicated by "NS" (not significant) in the table. Only Modifiability shows a significant correlation between its perceived business criticality and project success. In other words, *projects where modifiability is highly business critical tend to be less successful than projects where modifiability is less important.*

Table 4.3: Project success factors, correlation with business criticality of modifiability

Success Factor	**Kendall's τ**	**Sig. (1-tailed)**
Time	-.212	NS
Budget	-.219	NS
Efficient use of resources	-.207	NS
Customer satisfaction	-.324	.010
Solution quality	-.233	NS

Table 4.4: Cross-table of business criticality of modifiability and customer satisfaction

Count		Criticality modifiability				
		Very Low	Low	Medium	High	Very high
Customer satisfaction	Very Successful			3		1
	Successful	1	6	6	6	
	Neutral			6	4	
	Unsuccessful			1	2	
	Very Unsuccessful				1	2

Further analysis in Table 4.3 shows that this correlation can be attributed largely to one project success factor: customer satisfaction. This result is visualized in Table 4.4. The figure shows a remarkably consistent level of customer satisfaction for all projects where the architect judged business criticality of modifiability to be low or very low. As business criticality of modifiability grows, customer satisfaction ratings are spread over a wider range, and decrease on average.

4.3.2 Approaches and project success

The six requirements engineering approaches we consolidated from literature are expected to have a positive correlation with IT project success. For each identified approach, respondents had to indicate if it was applied and when it was applied during their latest completed project. The earlier the application of an approach in the systems development life cycle the higher the score, measured on a 6-point Likert-scale where each rating represents a project phase (requirements phase, design phase, realization phase, testing phase, deployment phase, later/never). The rationale behind this argument was described earlier. Statistical techniques are used to test the hypotheses and the results are presented in this section.

A summary of the results is presented in Table 4.5.

As seen from the table, only applying verification is positively correlated with IT project success.

Table 4.5: NFR Approaches and their correlation coefficient with IT project success

NFR Approach	**Kendall's τ**	**Sig. (1-tailed)**
Elicitation	.054	NS
Documentation	.065	NS
Quantification	.024	NS
Prioritization	.057	NS
Conflict analysis	-.128	NS
Verification	.256	.014

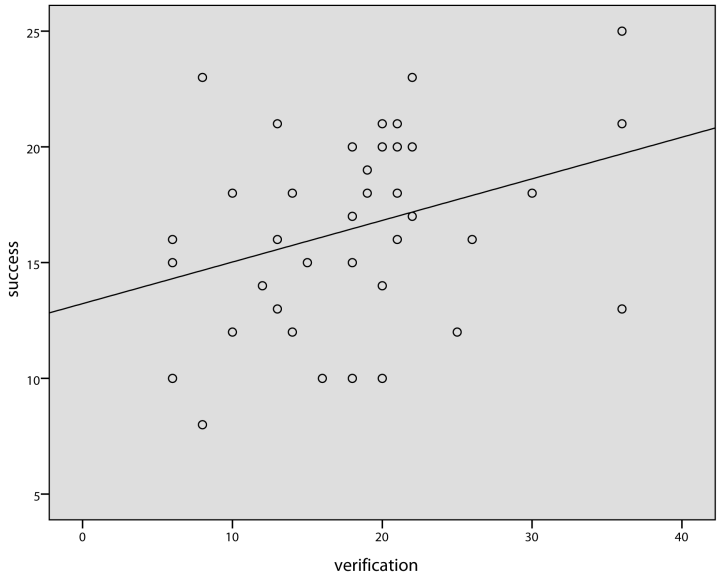


Figure 4.4: Plot of the correlation between timing of verification (higher score corresponds to earlier application) and project success

The correlation between verification and project success is visualized in Fig. 4.4. The horizontal axis in this figure represents a score based on when verification was applied, accumulated for all NFRs listed in §4.2.1: the higher the score, the earlier in the project verification was applied. There is a significant positive relationship between applying verification and IT project success, $\tau = .256$, p (one-tailed) $< .05$. In other words, we find that *projects where NFRs are verified in an early stage tend to be more successful than projects where NFRs are not verified or only at a later stage in the project.* This is not surprising; it is more surprising that we did not find such a correlation for any of the other approaches, which will be discussed further in the next section.

4.4 Discussion and Related Work

In this section, we further discuss the results found above, and share the key contributions from the post-survey analysis expert workshop. We will also discuss threats to validity, and relate our work to additional material found in literature.

4.4.1 Availability most business critical

In the perception of the architects responding to the survey , on average the business criticality of availability is highest. Earlier studies found similar results. For instance, in [Johansson et al., 2001] reliability was identified as the most important type of NFR in software platform development. Furthermore, in [Leung, 2001] reliability was ranked as the most important NFR and availability was ranked as the most important sub-characteristic for intranet applications. These studies used the six quality characteristics from the ISO/IEC 9126 standard as types of NFRs, where availability is a sub-characteristic of reliability. Furthermore, their definition of reliability is very similar to the definition of availability used in this research.

4.4.2 Non-functional requirements and project success

The results show that the perceived business criticality of modifiability is negatively correlated with IT project success. In other words: on average, IT projects where modifiability is seen as relatively important are significantly less successful than IT projects where modifiability is considered to be relatively unimportant. This correlation is largely due to the level of customer satisfaction.

The following three possible explanations for this phenomenon were generated by the post-survey workshop with architecture experts:

1. A high demand for modifiability might be an indication that the *customer does not know what he wants*. This means that a customer that demands high modifiability, is a customer that is more likely to change his requirements later on. A development team is trying to hit a moving target in such a situation. This explanation is in line with the leading role of customer satisfaction in the correlation.

2. Modifiability *leads to complexity*. Known techniques to realize high modifiability (such as layering, late binding and parameterizing) quickly lead to increasing complexity, with an adverse effect on budget and timescale. If this were the case, projects where modifiability is highly business critical would be expected not only to be less successful, but also larger and more prone to budget and schedule overruns. Thus, one would expect significant correlations between modifiability and project size, time and budget success factors. None of these correlations were found; in fact, some of the respondents that indicated low criticality for modifiability were working in some of the larger projects compared to other respondents. Thus, the survey yields no evidence supporting this theory.

3. Modifiability gets *too little attention*. This explanation appears to be confirmed by the relatively low scoring of modifiability in terms of perceived business criticality and application of techniques reported above. Expert workshop members experienced multiple reasons for "underappreciation" of modifiability:

    - modifiability is harder to quantify or measure, less "mathematical" than other NFRs; even though there are well known modifiability related code analysis metrics like cyclomatic complexity [McCabe, 1976], such metrics are seen as only indirectly related to the actual modifiability business goals, and easily "cheated"
    - other NFRs have a more direct effect on the project's business stakeholders (end-users, managers), while modifiability is sometimes perceived to become important only after the project is over - a dangerous view in light of the research presented here

No correlation is found between the business criticality of the other types of NFRs (availability, performance, security, usability and testability) and IT project success. This can either mean that the negative impact of NFRs is too small to be measured in a population this size, or that the dampening effect discussed before is in play: architects can only respond that NFRs are highly business critical if they are *aware* of this business criticality at the time of the survey. If an architect is aware of an NFR's business criticality at the time of creating the architecture, this awareness normally leads to addressing of the NFR in the architecture, thus reducing the risk to project success.

The expert workshop produced anecdotal evidence confirming the second theory. For example, Logica has a project unit that is specialized in highly reliable system construction. Projects where availability is highly business critical get assigned to this unit. This leads to economies of learning and thus more successful projects.

All this leads to the following conclusion regarding the link between NFRs and project success:

As long as the architect is aware of the business criticality 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.

4.4.3 Approaches and project success

The application of verification is positively correlated with IT project success. More specifically: IT projects that apply verification early in the development life cycle are significantly more successful than IT projects that apply verification late in the development life cycle. Verification was defined earlier as: verifying that a system fulfills NFRs, e.g. by prototyping, simulation, analysis, testing or other means. Although it is quite trivial that verification techniques reduce errors, there are apparently obstacles that prevent early verification of NFRs. This result indicates that practitioners should spend effort to overcome those obstacles.

It is surprising that none of the other approaches were found to have a significant effect on project success. After all, to be able to apply verification, shouldn't one at least have elicited and quantified the NFRs first? When evaluating the operationalization of the questions, some limitations come to mind. First, it might be more meaningful to measure *how* a certain approach was applied instead of measuring *when* it was applied. In the current situation, IT projects that very carefully elicited NFRs with multiple stakeholders using a formal method are not necessarily discriminated from IT projects where elicitation is informally applied in an ad-hoc fashion by a single stakeholder; moreover, the approaches are not really orthogonal with respect to the development phases. Second, the 6-point Likert-scale used is based on a general waterfall systems development life cycle and does not map very well unto iterative development methodologies. During the validation session, the experts judged that they were sufficiently aligned with the majority of the projects carried out by Logica. However, at least one respondent had trouble answering the questions about the application of the approaches, because his projects always use iterative development. These limitations mean we have to be careful interpreting this result, beyond that it is good to have some statistical evidence that early NFR verification is correlated with successful projects in at least one company.

4.4.4 Threats to validity and opportunities for further research

A few important limitations of this survey have to do with generalizability. First, the context of the research is architecture, since it has such a strong link with dealing with NFRs. This was a conscious choice, but it does mean that all results are subject to the perception of the projects' architects. It would be interesting to also investigate the impact of NFRs from other perspectives and compare the results. In particular, a study that would be able to distinguish between NFRs' business criticality and the architect's awareness of that criticality might shed more light on the material.

Second, the data was collected using respondents from a single organization. A cross-organizational approach would have been preferred, but this was not feasible due to practical limitations. Strictly speaking, the results are valid only in the context of Logica. However, Logica has many similarities with other similar companies. Moreover, over half of the ACoP architects (see §7.1) fulfil their roles on-site in customer organizations; so the results represent a mix of experiences in Logica and its customer base in the government, utilities, financial and other industrial sectors. Nevertheless, some results could be specific to Logica, and cannot be generalized without further research.

The measurement of the applied approaches was already mentioned as a limitation of this study. This could be a reason why no significant relationships were found between applying the approaches and IT project success except for verification. A study that focuses on measuring maturity of the applied approaches might be better capable to differentiate successful IT projects from unsuccessful ones. Another recommendation for future research would be to use a different kind of measurement for project success, e.g. including the actual customer and his evaluation of a project's success.

Other suggested extensions to future versions of this research are:

- Extend the definition of business criticality (see §4.2.1) to the company developing the software, rather than only its customers, which might yield a more balanced view on e.g. testability.

- Include *Designing for NFRs* in the list of approaches; this key activity of architects is left implicit in this survey, but making it explicit may yield additional interesting results.

- Ask the architects *when* they became aware of the business criticality of NFRs, to validate the conclusion at the end of §4.4.2.

4.5 Conclusions

We set out on this survey with the goal to investigate the awareness and handling of non-functional requirements among architects, and their effect on IT project success (research question RQ-1c).

The first part focused on trying to identify if certain types of NFRs have a relationship with IT project success. In other words, are there under-performing IT projects based on the types of NFRs they deal with? A significant negative relationship between the business criticality of modifiability and IT project success was found. Therefore, it can be concluded that IT projects where modifiability is relatively business critical perform significantly worse on average. Even though this result might be local to Logica, it provides a warning to all practitioners dealing with IT projects with a strong focus on modifiability. Aspects like quantification, verification and managing customer expectations around modifiability might require additional attention, because it seems that customer satisfaction especially is significantly lower on average in this type of IT projects.

The second part views the research question from another perspective: do approaches for dealing with NFRs have a positive influence on IT project success? From the results it can be concluded that the application of verification (starting as early as possible during the software development life cycle) has a positive influence on IT project success. In other words: IT projects that applied verification techniques relatively early in development were more successful on average, than IT projects that did not apply verification techniques (or applied it relatively late in development). As said earlier, practitioners should be aware that the long term benefits of verification outweigh the short term extra costs.

Part II

Establishing a Solution Architecting Approach

5

Case Study: Successful Architecture for Short Message Service Center

In Part I of this thesis, we examined how non-functional requirements be handled to improve the success of IT solutions and the projects delivering them (research question RQ-1). In Part II, we will now turn our attention to research question RQ-2: establishing a solution architecting approach to improve an IT service provider's success. This chapter starts off Part II with a small case study. It presents and analyzes the key architectural decisions in the design of a successful Short Message Service Center as part of a GSM network.

5.1 Introduction and Requirements

In the early nineties, a Short Message Service Center[1] was developed according to the specifications for text messaging embedded in the GSM standard [ETSI, 1995]. This chapter looks back at the conceptual design phase of the realization project. The chapter is a practitioner's report, analyzing the key architectural decisions and distinguishing factors that contributed to the system's success.

The SMSC's key requirements are listed below, according to the categorization presented in §2.3.2: first the primary (functional) requirements, and then the supplementary requirements, divided in secondary functional requirements and quality attribute requirements.

[1] This system was developed and commercially deployed by CMG, later LogicaCMG Telecoms and now Acision. In order to protect the commercial interests of the manufacturer, the descriptions have been left at a reasonably high level of abstraction, and data are mostly not quantified.

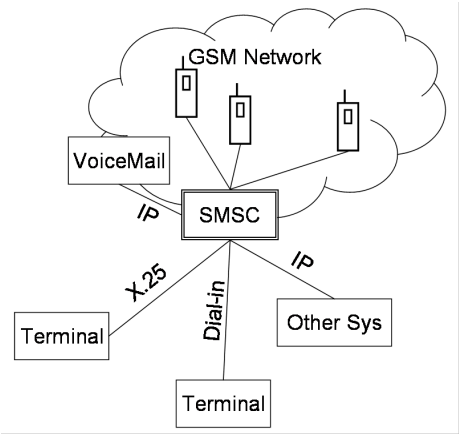


Figure 5.1: SMSC context.

Primary functional requirements

Fig. 5.1 shows the SMSC system in its primary context. The main purpose of the system is [PF1:] to pass messages between mobile telephones in a GSM network , and from and to other systems [PF2:] outside of the GSM network. Messages that cannot be immediately delivered are [PF3:] temporarily stored in the system.

Supplementary requirements

According to the classification in §2.3.2, supplementary requirements consist of three categories: Secondary Functional (SF) requirements, Quality Attribute (QA) requirements and Delivery requirements. For the purposes of this chapter, we only need to discuss the key SF and QA requirements.

The major secondary functional requirements were that [SF1:] a record of every message that has passed through the system is kept for billing purposes, and [SF2:] there is an interface to monitor and operate the system.

The major quality requirements set by the customers centered around [QA1:] *performance* of message throughput, [QA2:] *availability* of the messaging service and [QA3:] *reliability* of message storage. [QA4:] *Timeliness* in responses to external systems was critical. With a view to productizing of the solution, the manufacturer added requirements for [QA5:] *extensibility* and [QA6:] *scalability* of the solution.

5.2 Key Architectural Design Decisions

In order to fulfill the requirements set out above, the architects made some design choices that distinguished the system from other similar systems in three major aspects: *platform choice*, *storage strategy* and *interprocess communication*.

5.2.1 Platform choice

The main choice to be made with respect to the platform for the SMSC was between a traditional "telecom switch" platform and an IT platform. Even though the telecom switch platforms were better rated in terms of performance [QA1], availability [QA2] and reliability [QA3], IT platforms were deemed superior in terms of extensibility at a reasonable cost [QA5].

At the time of the design of the SMSC, the most popular platforms for these kinds of medium-high performance requirements were Unix environments. The development team, however, also had ample experience with OpenVMS platforms. It was felt that the OpenVMS platform would better be able to fulfill the timing requirements [QA4].

5.2.2 Storage strategy

The performance of the system [QA1] was important and was perceived to become more important later on [QA6]. For this reason, it was decided to use a system where messages were stored in memory and on disk in parallel. The permanent message store mechanism is based on proprietary OpenVMS file I/O. If a more conventional storage strategy would have been used, such as an RDBMS, the added resource usage needed to perform the more complex file operations would have made it harder to fulfill the performance requirements [QA1]. Thus, the chosen storage strategy provided a better fit with the non-functional requirements.

5.2.3 Interprocess communication

A process architecture over multiple nodes was necessary to fulfill the performance and scalability requirements [QA1,QA6], resulting in a need for transparent communication between processes (IPC) running on different hardware units. It was felt that using the commercial-off-the-shelf IPC products available at the time would cause problems fulfilling the performance and flexibility [QA5] requirements. The team decided to develop a mean-and-lean transparent IPC itself. The resulting utility was christened VIQ (Virtual Interprocess Queue).

5.3 Conclusions and Discussion

In the years following delivery of the system to the first customers, demand for short message services grew spectacularly. In the race to keep up with this growing demand, performance and reliability turned out to be the main deciding factors. The product quickly became the world's leading SMS product in terms of number of subscribers being serviced.

The major lesson we learned from this success story is to *beware of fashion in solution architecture.* In the SMSC case, key architectural choices deviated from the prevailing "fashion" at that time, because analysis indicated that the more popular practices were not the best choices to fulfill the key requirements of performance, timeliness and reliability. The deviations turned out to be the key distinguishing factors in the architecture, that led to a success story.

Practicing architects in our experience are often under pressure from managers and customers to follow trends and fashions in system design. This phenomenon can partly be attributed to personal risk management behavior: it is hard to blame a manager for making a wrong decision if many others made the same wrong decision. We frequently encounter the term "best practice" to rationalize decisions that follow trends and fashions, often without a clear trade-off analysis as to why these practices are best for that particular situation. For this reason, we prefer the term "best fit practice" to the ubiquitous "best practice".

Methods like the Cost Benefit Analysis Method [Kazman et al., 2002] can help architects to present the benefits of their choices in an objective way. This can be especially helpful when arguing choices that go against prevailing "fashion". It should, however, be kept in mind that the previously mentioned "career risk management" argument for following trends and fashions is not necessarily invalid, and risk management related quality attributes can rightfully show up in architecture evaluations [Clements et al., 2002].

This case presents an example of the benefits of an environment where architects can argue their choices and priorities in an objective manner, and select practices that best fit those priorities, rather than follow fashion. In this case, such an environment led to a solution that was very successful in terms of the business goals identified in §1.3: particularly consistency in delivery and customer satisfaction. In subsequent chapters, we will explore a number of factors that can contribute to such an environment, leading up to the establishment of a successful solution architecting approach.

6

The Influence of CMMI on Establishing an Architecting Process

In 2006, we started out to create a generic architecting process for Logica. Since the company had set an objective to achieve Maturity Level 3 of the Capability Maturity Model Integration® (CMMI®), the process needed to comply with the relevant requirements set by the CMMI. This chapter presents the elicitation of such requirements, and the resulting set of requirements. It analyzes their potential impact on generic architecting processes found in literature. It turns out that CMMI 1.3 is much stronger in support of architecting activities than CMMI 1.1 (the version for which we have done this analysis previously), but a few possible improvements remain.

6.1 Introduction

The setting of this chapter is the establishment of an institutionalized architecting process in Logica. We had established that such a process would help control technical risks in projects and products. At about the same time, a company-wide objective had been set to achieve CMMI Maturity Level 3. This made it necessary to obtain insight into the requirements that architecting processes need to fulfill in order to comply with CMMI-DEV Maturity Level 3 [1]. The required analysis to obtain this insight was originally done using CMMI version 1.1 and published in [Poort et al., 2007]. This chapter updates the analysis to CMMI version 1.3. There are now three CMMI constellations: Development, Service and Acquisition. Our work pertains to CMMI for Development (CMMI-DEV) [CMMI Product Team, 2010].

[1] *CMMI-DEV Maturity Level 3* is mostly abbreviated to *CMMI Level 3* in the rest of this chapter

As references we have chosen two generic processes found in literature: Architecture Based Development [Bass and Kazman, 1999], because its scope is close to our purpose and because it represents one of the better known approaches to architecting in both industry and academia, and [Hofmeister et al., 2007], because their model represents the commonalities between five industrial approaches.

First, in §6.2 we will present the organizational context and scope of a generic architecting process. In §6.3, the CMMI process areas that are relevant to such an architecting process will be identified, and their requirements on architecting processes extracted. In §6.4 follows a discussion on the impact of the CMMI requirements on generic architecting processes found in literature, and on the coverage of architecting processes by CMMI. We will finish up with some conclusions and further work to be done.

6.2 Architecting Process Context and Scope

6.2.1 Organizational context

The organizational context of this study was described in Chapter 1. One of the company's Technical Board's activities is controlling technical risks in the various IT projects and products. It was felt that technical risk control could be enhanced by developing and institutionalizing a process that would provide guidance for making technical decisions: in short, an architecting process.

The Technical Board's decision to institute an architecting process coincided with the setting of a maturity objective by the company's executive management. Encouraged by benefits experienced through local CMMI driven process improvement, management set an objective to achieve CMMI Maturity Level 3 for relevant organizational units throughout the whole company. This meant that the architecting process to be developed would be subject to the requirements set by the CMMI.

6.2.2 Scoping an architecting process

The terms Architecture and Architecting are used in a great variety of meanings in the IT world. Rather than risking a non-converging discussion on the meaning of the terms, it was decided to scope the architecting process in terms of a set of business goals and usage scenarios. For the purposes of this chapter, a high-level summary is provided:

- *Business Goals* The business goals for the architecting process were established as *Consistency in Delivery*, *Risk Management*, *Customer Satisfaction* and *Knowledge Incorporation*.

- *Usage Scenarios* The process will be used for architecting activities in the following scenarios: *Responding to a Request for Proposal (RfP)*, *Software Development Project*, *System Integration Project*.

The business goals and usage scenarios were analyzed to determine the scope of the architecting process. Apart from literature and the existing experience of the authors, additional input for the analysis came from other stakeholders, specifically the company's sales community, quality assurance community and technical community, obtained in a workshop.

The most significant elements in the outcome of this analysis are listed below.

- Analysis of the business goals and experience indicates that *architectural decisions* are critical to the success of solutions. The process should therefore give guidance on how to identify and make architectural decisions. This matches requirements from CMMI about decision analysis and resolution, and with recent publications about the status of architectural decisions [Bosch, 2004, Tyree and Akerman, 2005, van der Ven et al., 2006].

- Many architectural decisions are made during the *sales* phase of projects; the architecting process has to facilitate that process.

- A certain level of *reviewing and control* has to be facilitated by the process. This is the convergence of the architecture assessment practices from literature [Obbink et al., 2002, Clements et al., 2002], and the responsibilities of the Technical Board to control technical risks. Not only are reviewing and control necessary parts of the process, it also has to be facilitated by a certain level of standardization in *documentation* of architectures.

- The involvement of architects in the *implementation* phase of solutions is essential in order to assure that the selected solution will be adequately implemented *conforming* to the architecture. The architecting process has to facilitate this.

- To contribute to the business goal of knowledge incorporation, the process should support a structure for organizational *learning from experiences*. Learning points may be both process-related (like good practices) and product-related (like good architectural constructs).

- The objective is to implement a process that gives guidance on aspects of architecting that are *not specific* to particular types of applications, e.g. not just software development, but also system integration, ERP implementations, and

Table 6.1: Scope of architecting process: high-level requirements.

rq.arch	Give guidance on architecting technical solutions.
rq.arch.decision	Give guidance on how to make architectural decisions.
rq.arch.sales	Facilitate solution shaping during the sales process.
rq.arch.doc	Standardize architectural documentation.
rq.arch.controls	Give guidance on architectural controls.
rq.arch.conform	Assure conformance with architecture during the implementation process.
rq.scalable	Be scalable over business unit sizes (20 - 2000) and project/programme sizes (80K€ - 500M€), and over a broad range of size and complexity of solutions.
rq.generic	Be flexible / generic to work in diverse applications.
rq.generic.tailoring	Be accompanied by a set of tailoring guidelines.
rq.accessible	Be simple, accessible to all.
rq.accessible.terminology	Use terminology familiar to company staff.
rq.cmmi	Be CMMI Maturity Level 3 compliant.
rq.learning.product	Bottle product experiences and make them available to architects in a controlled manner.
rq.learning.process	Support a structure for organizational process learning.

embedded system development. This means its concept of "architecture" covers not only software, but the wider scope of *solution architecture*. For such a generic process to be useable, it must be accompanied by a set of guidelines for *tailoring* the process to the specific needs and characteristics of the usage environment. This is also required by CMMI Generic Practice 3.1 "Establish a Defined Process".

In summary, we need an architecting process description that focuses on requirements analysis, architectural decision making, shaping, selection and evaluation of the best-fit solutions, documenting and implementing architectures and controls like architectural governance and reviewing.

The scope of what is meant by an "architecting process" in this chapter is documented as a list of requirements[2] in Table 6.1. In §6.4.1, we will identify a number of generic architecting processes in literature that are similar in scope.

[2] A note on the tagging of requirements in this chapter: the reader will notice the use of mnemonic, hierarchical tagging [Gilb, 2005]. The use of dots indicates a hierarchical grouping, with an implicit traceability to higher level requirements.

The scope of the architecting process has been determined by the analysis of the business goals and usage scenarios, with limited consideration of CMMI. We will now focus on the influence of CMMI in more detail.

6.3 Architecting and CMMI

The Capability Maturity Model Integration (CMMI) is a process-improvement model developed by the Software Engineering Institute (SEI) of the Carnegie Mellon University. It is scoped towards the development, acquisition and maintenance of systems or services. CMMI-DEV is the CMMI constellation intended for solution development.

The "staged representation" of the CMMI-DEV consists of five maturity levels. With increasing maturity level, the process capabilities increase, resulting in a higher probability that development or maintenance targets will be realized [Gibson et al., 2006]. Each maturity level consists of a number of process areas (PAs). Each process area consists of a small set of *goals* followed by a collection of *practices* to be performed in order to realize the goals. In order to satisfy a process area, an organization must have visibly implemented the achievement of the process area *goals* in its processes. Before goals can be considered to be satisfied, either the process area *practices* as described, or acceptable alternatives to them, must be present in the planned and implemented processes of the organization. [CMMI Product Team, 2010] contains the goals and practices for all process areas, accompanied by information to help CMMI users understand them.

A process complies to a certain maturity level if the goals and practices of all process areas of that level are satisfied. The process areas are customarily referred to by a set of fixed tags; all level 2 and 3 process areas and their tags are listed in Table 6.2.

Goals and practices of a process area are divided into specific ones and generic ones. Specific goals and practices directly refer to the process area itself, whereas generic goals and practices represent mechanisms to institutionalize the specific goals and practices. These practices are called generic because they apply to multiple process areas.

CMMI Maturity Level 3 requires that for all process areas belonging to Level 2 and Level 3 a "defined process" is established. A defined process is tailored from the organization's "standard process" according to a set of tailoring guidelines. In addition, a defined process has a maintained process description, which implies that all (generic and specific) practices are described. For more information, the reader is referred to [CMMI Product Team, 2010].

This section starts with an exploration of what a *CMMI Compliant Architecting Process* actually means. This is followed by a discussion on the use of architectural

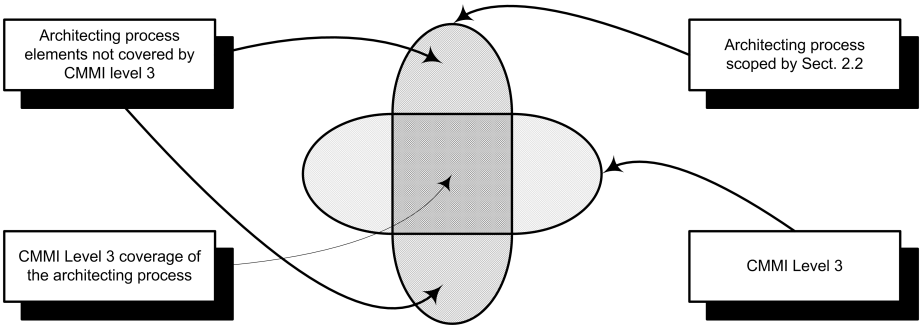


Figure 6.1: CMMI coverage of the architecting process.

concepts in the CMMI. We then proceed to identify the process areas that have a significant contribution to architecting according to the scope set out in §6.2.2. We call this set the *architecting significant process areas (ASPAs)*.

6.3.1 CMMI-compliant architecting process

The boundaries (scope) of the architecting process are determined in §6.2.2. Because of the structure of the CMMI, the practices related to this process may be distributed over a number of process areas.

The CMMI Level 3 coverage of the architecting process can be obtained by analyzing every Level 2 and Level 3 specific practice to determine whether or not the practice is inside the scope of the architecting process. The generic practices of Level 2 and Level 3 will always be in scope because they apply to all process areas. This analysis will be performed further on in this chapter.

Fig. 6.1 illustrates the CMMI coverage of the architecting process. As can be derived from the figure, the architecting process may include elements that are not covered by CMMI Level 3. These may for example be elements that are beyond the scope of system development (like architectural roadmapping) or elements that are considered critical for a successful architecting process but cannot be found in the CMMI.

Summarizing the above information, it can be stated that a CMMI Level 3 compliant architecting process:

- has a maintained description of all specific and generic practices that are in scope of the architecting process (the square box in the figure)

- has a maintained description of guidelines to tailor the process to the specific needs and characteristics of the usage environment
- is consistently deployed inside the company in the context of the user scenarios referred to in §6.2.2.

The scope of this chapter is the determination of the practices that should be part of the maintained description mentioned in the first two items. These practices will be presented as a list of requirements imposed on an architecting process description. In §6.3.3 we will present the elicitation of these requirements, but first we will have a more general look at the use of architecture concepts in the CMMI.

6.3.2 Architecture concepts in the CMMI

The word "architecture" is used extensively in the CMMI. It appears in 12 out of 22 process area descriptions [CMMI Product Team, 2010]. The CMMI is a collection of industry best practices and not a formal theoretical model. Effort was put in making the model consistent and unambiguous, but many parts are still subject to different interpretations.

Architecture itself is defined in the CMMI-DEV 1.3 glossary. Apart from its defined usage, the word is also used in the concept of "process architecture" to denote designing of company processes. This type of activity is outside the scope of this chapter as defined in §6.2.2, and we have filtered out this usage in our analysis.

Several architecture-related terms are defined in the CMMI glossary:

- *Architecture* is defined as: "The set of structures needed to reason about a product. These structures are comprised of elements, relations among them, and properties of both." The glossary explicitly points out the role of quality attributes in the context of architecture. This definition is quite close to our definition of solution architecture in §1.2.1 on page 3, except that it lacks the keyword "fundamental".
- *Functional architecture* is defined as: "The hierarchical arrangement of functions, their internal and external (external to the aggregation itself) functional interfaces and external physical interfaces, their respective requirements, and their design constraints"
- *Definition of required functionality and quality attributes*: "A characterization of required functionality and quality attributes obtained through "chunking," organizing, annotating, structuring, or formalizing the requirements (functional and

non-functional) to facilitate further refinement and reasoning about the requirements as well as (possibly, initial) solution exploration, definition, and evaluation." This term refers to the beginning of architecting activities, and is within the scope of our architecture process.

- *Nontechnical requirements* are defined as: "Requirements affecting product and service acquisition or development that are not properties of the product or service." This term coincides with our definition of "Delivery requirements" in §2.3.2 (page 19).

- *Quality attribute* is defined as: "A property of a product or service by which its quality will be judged by relevant stakeholders. Quality attributes are characterizable by some appropriate measure." The CMMI glossary explicitly links quality attributes to architecture.

- *Shared Vision* is defined as: "A common understanding of guiding principles, including mission, objectives, expected behavior, values, and final outcomes, which are developed and used by a project or work group."

One other architecture-related term is used extensively, but not defined: *design*. Since a design is definitely a structure needed to reason about a product, one could argue that it falls under the CMMI definition of architecture. We include guidance about design in CMMI-DEV in our analysis wherever it falls within our scope as defined in §6.2.2.

These considerations show that a number of key concepts and terms relevant to architecting are defined in the CMMI. The following section will discuss the identification of the CMMI requirements on an architecting process.

6.3.3 Process areas relevant to architecting

Our approach to establish which requirements CMMI imposes on architecting processes is to first identify which process areas are relevant for the process, and then to extract requirements on the process from the practices in their descriptions. An analysis of the CMMI Level 3 process areas against the architecting process scoped in §6.2.2 results in a set of process areas that have a direct and significant contribution to the objectives of this process. As discussed before, these process areas are called *architecting significant process areas* (ASPAs).

The process areas of the CMMI are grouped into four categories:

- *Process Management*. These process areas contain the activities related to defining, planning, implementing, monitoring, evaluating and improving all other processes. The architecting process is subject to these process management process areas in order to assure the required level of capability.

- *Project Management*. These process areas cover the project management activities related to planning, monitoring and controlling the development or maintenance project. The architecting process is generally performed in the context of a project.

- *Engineering*. These process areas cover the development and maintenance activities that are shared across engineering disciplines (e.g. systems engineering and software engineering). The architecting process falls mainly within these process areas.

- *Support*. These process areas cover the activities that support all other process areas like establishing measurement programs, verification of compliance, and effective decision making. The architecting process is also subject to these process areas.

Table 6.2 identifies the categorized set of Level 3 process areas and indicates which process areas have been qualified as an architecting significant process area. It should be noted that *all* process areas of the CMMI contribute to the objectives of the architecting process. Their contribution may be direct because the process area is actually part of the architecting process, or indirect because the process area is establishing the context and preconditions for a successful architecting process.

As stated before an architecting significant process area has a direct contribution and this contribution should also be significant. This is the case for all Engineering process areas, two Project Management process area (Risk Management and Requirements Management) and one Support process area (Decision Analysis and Resolution). Risk Management, Requirements Management and Decision Analysis and Resolution are actually part of the architecting process and contribute significantly to its objectives. The architecting relevance of the set of architecting significant process areas is shortly explained below. Where relevant, underpinning references to the CMMI text have been added in [braces].

REQM *Requirements Management.* The role of architecting in Requirements Management focuses around the impact of requirements and their traceability to the architecture. [Specific Practice (SP)1.1 *Understand Requirements* describes the process of the acceptance of requirements according to objective criteria. “Consistent with

Table 6.2: Categorized Level 2 & 3 Process areas and their architecting significance.

Tag		ASPA
Process Management		
OPF	Organizational Process Focus	
OPD	Organizational Process Definition	
OT	Organizational Training	
Project Management		
PP	Project Planning	
PMC	Project Monitoring and Control	
SAM	Supplier Agreement Management	
IPM	Integrated Project Management	
RSKM	Risk Management	Y
REQM	Requirements Management	Y
Engineering		
RD	Requirements Development	Y
TS	Technical Solution	Y
PI	Product Integration	Y
VER	Verification	Y
VAL	Validation	Y
Support		
CM	Configuration Management	
PPQA	Process and Product Quality Assurance	
MA	Measurement and Analysis	
DAR	Decision Analysis and Resolution	Y

architectural approach and quality attribute priorities" is an example criterion relevant to architecting. It is also relevant to the impact analysis mentioned in SP1.3 *Manage Requirements Changes*: "Requirements changes that affect the product architecture can affect many stakeholders." SP1.4 *Maintain Bidirectional Traceability of Requirements*: traceability to architectural components is mentioned: "Work products for which traceability may be maintained include the architecture, product components, development iterations (or increments), functions, interfaces, objects, people, processes, and other work products.". Traceability to architectural decisions is implied.]

RD *Requirements Development.* This process area is where functional and quality attributes requirements are elicited, analyzed and established. Architecting is important here both as a source of new requirements and as a means to structure requirements. ["Analyses occur recursively at successively more detailed layers of a product's architecture". Specific Goal 2 *Develop Product Requirements* identifies the selected product architecture as a source of derived requirements. SP2.1 *Establish Product and Product-Component Requirements* prescribes to "develop architectural requirements capturing critical quality attributes and quality attribute measures necessary for establishing the product architecture and design". SP2.3 *Identify Interface Requirements* prescribes the definition of interfaces as an integral part of the architecture definition. SP3.2 *Establish a Definition of Required Functionality and Quality Attributes* prescribes the identification of architecturally significant quality attributes. Other important architecting activities are impact and risk assessment of the requirements, mentioned under SP3.4 Analysis and SP3.5 Validation.]

TS *Technical Solution.* This process area covers the core of architecting: developing a solution that fulfills the requirements. [TS specific goals are SG1 *Select Product Component Solutions*, SG2 *Develop the Design* and SG3 *Implement the Product Design*. SP1.1 *Develop Detailed Alternative Solutions and Selection Criteria* prepares architectural decision making by identifying alternatives and selection criteria. SP1.2 *Select Product Component Solutions* and SP2.4 *Perform Make, Buy or Reuse Analyses* are about making design decisions and documenting them, including rationale. SP2.1 *Design the Product or Product Component* establishes the product architecture. It describes architecture definition, driven by the architectural requirements developed in RD SP 2.1. It identifies elements of architectures, such as coordination mechanisms, structural elements, standards and design rules. It also mentions architecture evaluations to be conducted periodically throughout product design. SP2.2 *Establish a technical data package* gives guidance on where the architecture definition and the rationale for key decisions are documented. SP2.3 *Design Interfaces Using Criteria* supplies requirements to the interface design process.]

PI *Product Integration.* In this process area, the architecture is implemented in an actual integrated system and delivered. The architecting significance of the process area lies in the involvement of the architect in the implementation phase, and the architectural significance of the integration strategy [Product Integration has three Specific Goals (SG): *Prepare for Product Integration*, *Ensure Interface Compatibility* and *Assemble Product Components and Deliver the Product*. These goals should be achieved in line with the product architecture.

VER *Verification.* Verification is an essential part of the architecting process because its purpose is to ensure that the work products of this process meet the specified requirements. Typical work products of the architecting process are the architecture and design documents and the architecture and design itself. Means for verification may be peer reviews (for documents) and architectural assessments. Verification activities should be prepared, performed, the results analyzed and corrective actions identified.

VAL *Validation.* Validation is in fact a variant on verification but its objective is to demonstrate that a (work) product fulfills its intended use when placed in its intended environment (i.e. that it meets user needs). Regarding the architecting process, the work products and means for validation are similar to verification.

DAR *Decision Analysis and Resolution.* Key to architecting is decision making [Bosch, 2004, Tyree and Akerman, 2005]. The DAR process area prescribes a formal evaluation process for decisions of this kind: evaluation criteria should be established, alternatives should be identified, evaluation methods selected, alternatives evaluated and a solution selected. There should also be guidelines establishing which decisions should be subject to this formal evaluation process. Many DAR requirements overlap with selection practices in Technical Solution. ["When competing quality attribute requirements would result in significantly different alternative architectures." is listed as a typical guideline for requiring formal evaluation.]

RSKM *Risk Management*. Better risk management is one of the business goals of the architecting process. The inherent risk in a requirement is an important factor in determining whether or not it is an architectural requirement, as will be explained in Chapter 8. [A requirement that, when not fulfilled, heavily "affects the ability of the project to meet its objectives" (SP1.1 Determine Risk Sources and Categories), has a good chance to be considered architectural. Typical architectural risk sources listed are "uncertain requirements" and "Competing quality attribute requirements that affect solution selection and design". The RSKM process area prescribes how to deal with such risks: risk parameters should be defined (SP1.2), a risk management strategy should be

established (SP1.3), the process should give guidance on how risks are identified and analyzed (SG2), and mitigated (SG3). Insofar as architectural requirements involve risks, they should be treated the same way.]

An analysis of the texts of these architecting significant process areas yields the requirements imposed on the architecting process by the CMMI. These requirements are listed in Table 6.3. In agreement with the nature of the CMMI, this table is effectively a list of 73 best practices that support companies in creating and implementing an architecting process.[3]. These best practices are based on the *informative* text accompanying the architecting significant process areas in [CMMI Product Team, 2010], so strictly speaking an architecting process that does not fulfill the requirements can still be CMMI compliant, as long as the architecting significant process area goals are visibly fulfilled by alternative practices. For the purposes of this analysis, however, we have based the requirements on the architecting significant process area texts. The tags in Table 6.3 allow traceability to the process areas that the requirements originated from, and give the list a clear structure. The largest contributor is Technical Solution (TS) with 30 requirements, confirming our earlier observation that TS covers the core of architecting. The next largest contributor is Requirements Development (RD) with 21 requirements, indicating that an architecting process within our scope includes a substantial amount of requirements development practices. All other process areas provide only 4 or less requirements.

6.4 Discussion

In this section, we will discuss our results in conjunction with two generic architecting process models found in literature, and we will discuss the coverage of architecting processes in CMMI.

6.4.1 Generic architecting process models in literature

The CMMI imposes requirements on processes used by organizations. So if an organization were to institutionalize an architecting process based on a model found in literature, what would that organization have to do to make their architecting process CMMI level 3 compliant?

Although this analysis of CMMI's influence on architecting processes was based on an initial scope set out in the context of a particular company setting, the results of

[3]CMMI version 1.1 yielded 67 [Poort et al., 2007]), giving a quantitative indication of the improved support for architecting in version 1.3

Table 6.3: Requirements imposed on Architecting Process by CMMI.

rq.cmmi.reqm.arch	Use architectural fit as criterion when assessing requirements and changes.
rq.cmmi.reqm.trace	Maintain traceability between requirements and architectural components and decisions.
rq.cmmi.rd.doc	Translate stakeholder needs, expectations, constraints, and interfaces into documented customer requirements.
rq.cmmi.rd.prio	Establish and maintain a prioritization of customer functional and quality attribute requirements.
rq.cmmi.rd.fun-arch	Develop a functional architecture.
rq.cmmi.rd.recursive	Analyze requirements recursively.
rq.cmmi.rd.arch-req	Develop architectural requirements capturing critical quality attributes necessary for establishing architecture.
rq.cmmi.rd.tech	Develop requirements in technical terms necessary for product and product component design.
rq.cmmi.rd.drivers	Determine key mission and business drivers, and determine architecturally significant quality attributes based on them.
rq.cmmi.rd.part	Partition requirements into groups, based on established criteria, to facilitate and focus the requirements analysis.
rq.cmmi.rd.alloc	Allocate requirements and design constraints to product components and the architecture, and to functional elements.
rq.cmmi.rd.derive	Derive requirements that result from design decisions.
rq.cmmi.rd.if	Identify interface requirements.
rq.cmmi.rd.depend	Establish and maintain relationships between requirements.
rq.cmmi.rd.an	Analyze requirements.
rq.cmmi.rd.an.key	Identify key requirements that have a strong influence on cost, schedule, performance, or risk.
rq.cmmi.rd.an.perf	Identify technical performance measures that will be tracked during the development effort.
rq.cmmi.rd.an.scen	Develop and analyze operational concepts and scenarios.
rq.cmmi.rd.balance	Use proven models, simulations, and prototyping to analyze the balance of stakeholder needs and constraints.
rq.cmmi.rd.risk	Perform a risk assessment on the requirements and definition of required functionality and quality attributes.
rq.cmmi.rd.impact	Assess the impact of architecturally significant quality attribute requirements on product and development costs and risks.
rq.cmmi.rd.lifecycle	Examine product life-cycle concepts for impacts of requirements on risks.
rq.cmmi.rd.assess	Assess the design as it matures in the context of the requirements validation environment.
rq.cmmi.ts.alt	Develop detailed alternative solutions to address architectural requirements.
rq.cmmi.ts.alt.cots	Identify candidate COTS products that satisfy the requirements.
rq.cmmi.ts.alt.techn	Identify technologies currently in use and new product technologies for competitive advantage.
rq.cmmi.ts.alt.reuse	Identify re-usable solution components or applicable architecture patterns.
rq.cmmi.ts.alt.crit	Develop the criteria for selecting the best alternative solution, typically addressing costs, schedule, benefits and risks.
rq.cmmi.ts.alt.crit.eval	Based on the evaluation of alternatives, assess the adequacy of the selection criteria and update them as necessary.
rq.cmmi.ts.alt.issues	Identify and resolve issues with the alternative solutions and requirements.
rq.cmmi.ts.alt.eval	Evaluate alternative solutions against criteria.
rq.cmmi.ts.alt.acq	Identify the product component solutions that will be reused or acquired.
rq.cmmi.ts.alt.doc	Establish and maintain the documentation of the solutions, evaluations, and rationale.
rq.cmmi.ts.scenario	Evolve operational concepts and scenarios.
rq.cmmi.ts.design	Establish the product architectural design.
rq.cmmi.ts.design.struct	Establish product partition into components.
rq.cmmi.ts.design.struct.if	Identify and document major intercomponent interfaces.
rq.cmmi.ts.design.struct.id	Establish product-component and interface identifications.
rq.cmmi.ts.design.state	Establish main system states and modes.
rq.cmmi.ts.design.if	Identify and document major external interfaces.
rq.cmmi.ts.design.crit	Establish and maintain criteria against which the design can be evaluated.
rq.cmmi.ts.design.method	Identify, develop, or acquire the design methods appropriate for the product.
rq.cmmi.ts.design.standard	Ensure that the design adheres to applicable design standards and criteria.
rq.cmmi.ts.design.fulfill	Ensure that the design adheres to allocated requirements.
rq.cmmi.ts.doc	Document and maintain the design in a technical data package.
rq.cmmi.ts.doc.levels	Determine the number of levels of design and the appropriate level of documentation for each design level.
rq.cmmi.ts.doc.views	Determine the views to be used to document the architecture.
rq.cmmi.ts.doc.impl	Base detailed design descriptions on the allocated product-component requirements, architecture, and higher level designs.
rq.cmmi.ts.doc.rationale	Document the key decisions made or defined, including their rationale.
rq.cmmi.ts.if	Establish and maintain interface descriptions.
rq.cmmi.ts.if.crit	Design interfaces using criteria.
rq.cmmi.ts.implement	Implement design adhering to design decisions and architecture.
rq.cmmi.pi.seq	Guidance on determining the product integration sequence.
rq.cmmi.pi.if	Ensure interface compatibility of product components, both internal and external.
rq.cmmi.pi.if.review	Review interface descriptions for completeness.
rq.cmmi.pi.if.manage	Manage interface definitions, designs, and changes.
rq.cmmi.ver.prepare	Prepare verification activities.
rq.cmmi.ver.peer	Perform peer reviews on architecture and design documents.
rq.cmmi.ver.eval	Perform architecture evaluations.
rq.cmmi.ver.conform	Verify architecture conformance of the implementation.
rq.cmmi.ver.analyze	Analyze verification results and identify corrective actions.
rq.cmmi.val.prepare	Prepare validation activities.
rq.cmmi.val.validate	Validate (part of) the architecture or design.
rq.cmmi.val.analyze	Analyze validation results and identify corrective actions.
rq.cmmi.dar.guid	Specify when a technical choice or design decision is architectural and subject to formal decision process.
rq.cmmi.dar.rank	Evaluation criteria for alternative solutions should be ranked.
rq.cmmi.dar.evalmethod	Guidance on selecting evaluation methods for alternatives.
rq.cmmi.rskm	Guidance on handling architectural requirements as risks.
rq.cmmi.rskm.id	Identify architectural risks.
rq.cmmi.rskm.analyze	Analyze architectural risks.
rq.cmmi.rskm.mitigate	Mitigate architectural risks.
rq.cmmi.gen	Architecting process should be institutionalized according to CMMI's Generic Practices.

the analysis should be relevant for other generic architecting processes. This section explores that relevance. We examine the impact of the CMMI requirements derived in this chapter on two generic architecture process models found in literature: one from a technical report and one from a journal paper. Please note that the architecting process models treated here differ significantly in scope: one focuses on design and analysis and the other focuses on architecture playing a central role throughout the software development lifecycle process. Also note that the models only roughly overlap the architecting process scope set out in §6.2.2.

Architecture-Based Development (ABD)

This is the generic architecting process as developed by the Architecture group at the SEI. It is described in [Bass and Kazman, 1999], but aspects of it are present in most of the publications of the SEI Architecture group (e.g. [Bass et al., 2003]). It is used as a reference here because its scope is close to that determined in §6.2.2, and because it represents one of the better known approaches to architecting in both industry and academia.

The ABD process consists of six activities:

1. Elicit the architectural requirements.
2. Design the architecture.
3. Document the architecture.
4. Analyze the architecture.
5. Realize the architecture.
6. Maintain the architecture.

Table 6.4 shows how the architecting significant process areas map onto these steps. In order to make the ABD process CMMI Level 3 compliant, each of these steps should be implemented in such a way that the practices belonging to the architecting significant process areas related to this step are satisfied. The following explanation applies to this mapping:

- Requirements Development (RD) is not only mapped onto the Elicit step but also onto the Design step. This is because the establishment of the "functional architectural structure" as part of this step is actually a practice that is part of RD.

- Verification (VER) activities start from the Design step because, as discussed before, verification refers to the requirements produced during the Elicit step.

Table 6.4: ASPAs Mapping onto ABD Steps.

	Elicit	Design	Document	Analyze	Realize	Maintain
REQM	X					
RD	X	X				
TS		X	X	X	X	X
PI					X	
VER		X	X	X	X	X
VAL	X	X	X	X	X	
RSKM	X	X	X	X	X	X
DAR	X	X	X	X	X	X

- The ABD process defines that each step includes validation (VAL) activities. For the Elicit step this refers to the validation of behavioral and quality scenarios.

- The Maintenance step is not well defined and scoped in the ABD process description. The existing text refers to means to prevent that the architecture drifts from its original precepts due to poor maintenance. This may include activities to extract the architecture of the as-built system, verify its level of compliance with the architecture of the as-designed system and performing the required corrective actions. In this respect, Technical Solution (TS) and Verification (VER) should be mapped onto the Maintenance step.

- Since Risk Management (RSKM) and Decision Analysis and Resolution (DAR) generally support all development and maintenance activities, they are related to all steps of the ABD process.

Generalized software architecture design model

[Hofmeister et al., 2007] compare five industrial approaches to architectural design, and extract from their commonalities a general software architecture design approach. The approach involves three activities:

1. *Architectural analysis*: define the problems the architecture must solve. This activity examines architectural concerns and context in order to come up with a set of Architecturally Significant Requirements (ASRs).

Table 6.5: ASPAs Mapping onto Generalized Architecture Design Model Activities.

	Analysis	Synthesis	Evaluation
REQM	X		
RD	X		
TS		X	
PI			
VER			X
VAL			
RSKM	X	X	X
DAR	X	X	X

2. *Architectural synthesis*: the core of architecture design. This activity proposes architecture solutions to a set of ASRs, thus it moves from the problem to the solution space.

3. *Architectural evaluation*: ensures that the architectural design decisions made are adequate. The candidate architectural solutions are measured against the ASRs.

It should be noted that this generalized model is of a higher level of abstraction than the ABD process discussed before, and that its scope excludes the realization of the architecture (it is design focused, as the name "generalized architecture design model" implies).

Table 6.5 shows how the selected set of architecting significant process areas map onto these activities. In order to make a process based on this generalized model CMMI Level 3 compliant, each of these activities should be implemented in such a way that the practices belonging to the architecting significant process areas related to this activity are satisfied. The following explanation applies to this mapping:

- Unlike the ABD process, the generalized model excludes architecture realization from its scope. For this reason, PI cannot be mapped to this model.

- The Architectural Evaluation activity ensures that the architectural design decisions made are adequate. The candidate architectural solutions are measured against the architecturally significant requirements (ASRs). Although the result is called the *validated* architecture, this activity is *verification* (VER) in CMMI terms because it refers to the requirements (ASRs) produced during the Archi-

tectural Analysis activity. *Validation* (VAL) in CMMI terms (against the user needs behind the requirements) is not part of the generalized model.

- Since Risk Management (RSKM) and Decision Analysis and Resolution (DAR) generally support all development and maintenance activities, they are related to all activities of the generalized model.

6.4.2 CMMI Coverage of architecting processes

As discussed in §6.3, the architecting process scoped in §6.2.2 may include elements that are not covered by CMMI Level 3. An analysis of the information in this section against the CMMI yields the following elements that are not or only indirectly covered.

rq.arch.doc Standardization of architectural documentation: the activity to document architecture and design information is part of the practices of Technical Solution (TS), including what kind of information should be documented and guidance on how it should be organized. In this way the CMMI guides standardization of documents. In CMMI 1.3, guidance on architecture documentation is significantly improved over CMMI 1.1, including e.g. the use of views [ISO 42010, 2011].

rq.arch.conform Facilitating conformance to architecture during the implementation process: The implementation phase as such is part of the practices of Technical Solution (TS) and Product Integration (PI), including references to Verification (VER) in order to verify the implementation once it is finished. CMMI 1.1 did not provide any explicit support in ensuring that the architecture and design will be adequately implemented during implementation; CMMI 1.3 gives more guidance, including the use of architecture evaluations and the role of quality attributes.

rq.learning.product Bottle experiences and make available for architects: the CMMI has many process areas that deal with establishing an infrastructure for organizational learning and improvement. Because the CMMI is a process framework, this is strongly focussed on the process dimension (like the architecting process), not on the product dimension (like architectural solutions). Only at Level 5 the process area Organizational Innovation and Deployment (OID) addresses improvements on processes and (process and product related) technologies. Product related technologies may also be interpreted as architectural solutions.

rq.arch.controls The requirements for controls like architectural governance and reviewing in CMMI is limited. Reviewing is covered in the Verification (VER)

process area. Architectural governance, however, is largely missing. With architectural governance we mean activities to manage architectural resources like reference or enterprise architectures, or the architects themselves. The only reference made to resourcing architects is an example in Generic Practice 2.4: "Appointing a lead or chief architect that oversees the technical solution and has authority over design decisions helps to maintain consistency in product design and evolution." The only type of architectural asset discussed is a software product line's core asset base. The lack of architecture-specific governance guidance is a logical consequence of the fact that the CMMI model places such governance activities in Generic Practices: they are abstracted away from specific application areas.

rq.arch.sales CMMI-DEV offers no support for the sales process. Some examples in [CMMI Product Team, 2010] refer to the existence of a contract, but the definition of Customer is limited to "The party responsible for accepting the product or for authorizing payment." As we have seen in Chapter 3, a sales process can have significant impact on architecting activities. CMMI could be improved by acknowledging this impact and giving guidance on it.

An informal visualization of the overlap between CMMI and the architecting process is presented in Fig. 6.2. In this figure, CMMI process areas (circles) and architecting process requirements (ovals) are plotted onto the areas of Fig. 6.1. Elements in the overlapping square area are covered by both CMMI and our architecting process scope. Partly covered elements are plotted straddling the scope boundary lines.

A note on the meaning of the fact that some elements are not covered by CMMI. We have not made any statement on the relative merits of these elements. One could argue that this lack of coverage is a shortcoming of CMMI; conversely, one could argue that, given the success of CMMI, how do we know that the elements that *are* covered by CMMI aren't by themselves good enough for an optimal architecting process? The current state of affairs does not allow us to answer this question in a general sense; the analysis in §6.3 merely indicates that in the current organizational setting, the elements would contribute to achieving the business goals set.

6.5 Conclusions and Further Work

Our starting point in this chapter was a large IT company with a need to institutionalize a generic architecting process that is compliant with CMMI Maturity Level 3. To this end, we have studied and discussed the relation between architecting and CMMI, resulting in the identification of process areas significant to architecting, and a list of

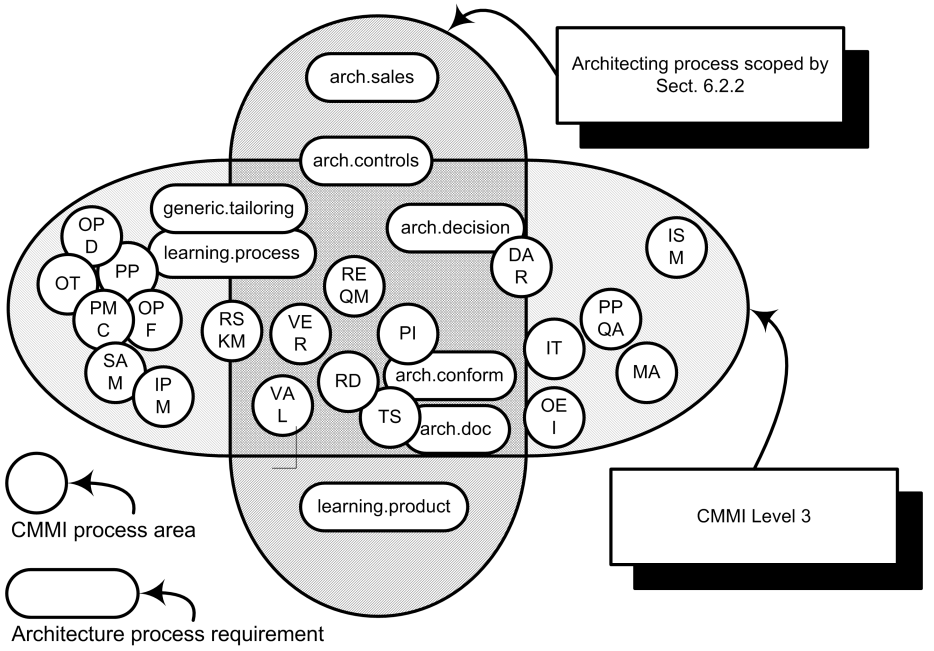


Figure 6.2: CMMI, architecting process and cross-section. See p.88 for PA abbreviations.

requirements to make a generic architecting process compliant with CMMI Maturity Level 3. Furthermore, we have compared our findings with two well-known process models from literature.

We conclude that:

- Architecture is a well-defined concept in the CMMI-DEV 1.3.
- CMMI-DEV 1.3 provides considerable support in establishing an architecting process. However, in some areas of architecting, the CMMI only gives weak support. The weaker areas are architecture governance, facilitating the sales phase, and learning from architectural choices.

Besides these conclusions, other relevant findings worth mentioning are:

- Although the scope of this chapter was limited to CMMI Level 3, an investigation of the level 4 and 5 process areas shows that none of these are Architecting Significant according to our scope, with the possible exception of Causal Analysis and Resolution.
- Although architecting is generally viewed as an engineering activity, three process areas outside Engineering are crucial to a good architecting process: Requirements Management, Risk Management and Decision Analysis and Resolution. This may not seem significant if the placement of these process areas outside of Engineering is merely seen as a structural choice in the CMMI model. Organizations should, however, not make the mistake of not applying these processes to engineering, or not involving their (architecting) engineers in them.
- CMMI 1.3 is much improved over version 1.1 in terms of support for architecting.

Further work

As has been mentioned in the introduction to this chapter, the work described here was done in the context of designing a generic architecting process for a large IT company. This work gave rise to the remaining chapters in this Part.

The generalized architecture design model discussed in §6.4.1 returns in Chapter 8, where we will use it to illustrate the impact of our insight into the nature of architecture.

An architecting process that complies with a maturity model also begs a comparison with Architecture Maturity Models (AMMs), such as the IT Architecture Capability Maturity Model (ACMM) developed by the US Department of Commerce [US Department of Commerce, 2007]. This comparison could be subject of a future analysis.

7

Successful Architectural Knowledge Sharing: Beware of Emotions

This chapter presents the analysis and key findings of a survey on architectural knowledge sharing. The responses of 97 architects working in the Dutch IT Industry were analyzed by correlating practices and challenges with project size and success. Impact mechanisms between project size, project success, and architectural knowledge sharing practices and challenges were deduced based on reasoning, experience and literature. We find that architects run into numerous and diverse challenges sharing architectural knowledge, but that the only challenges that have a significant impact are the emotional challenges related to interpersonal relationships. Thus, architects should be careful when dealing with emotions in knowledge sharing.

7.1 Introduction

In recent years, architectural knowledge (AK), including architecture design decisions, has become a topic of considerable research interest. Management and sharing of AK are considered to be important practices in good architecting [Lago and van Vliet, 2006, Tyree and Akerman, 2005, Clements and Shaw, 2006, Farenhorst and de Boer, 2009, Ali Babar et al., 2009]. In our quest to improve solution architecting, we decided to look into the relationship between architectural knowledge sharing and challenges in solution delivery projects.

In the beginning of 2008, the members of the Logica Netherlands architecture community of practice were surveyed. The main reason for this survey was to establish a baseline of current practice in architectural knowledge sharing (AKS), and to gain insight into the mechanisms around AKS and related challenges in projects. These ob-

jectives together amount to RQ-2c. The company was interested in these mechanisms because they saw architectural knowledge management as a way to improve IT project performance. The architects were asked about the content, manner, reasons and timing of the AK sharing they did in their latest project; both obtaining and sharing knowledge towards others. They were also asked about the challenges they faced. Furthermore, they were asked to identify various properties of their latest project's context, such as project size and success factors.

Even though the architects surveyed all work for the same IT services company, according to the survey 64% of them is doing so mostly at customers' sites. As a consequence, the survey results represent a mix of AK sharing practices in Logica and in Logica's customer base, which includes major Dutch companies and government institutions.

7.2 Survey Description

The invitation to participate in the survey was sent out by e-mail to 360 members of the Netherlands (NL) Architecture Community of Practice (ACoP) of the company. The ACoP consists of experienced professionals practicing architecture at various levels (business, enterprize, IT, software, and systems architecture) in project or consultancy assignments. The survey was closed after 3 weeks. By that time, 142 responses were collected. 97 respondents had answered the majority of the questions (93 had answered all). The other 45 responses were discarded because no questions about AK sharing had been answered. The survey consisted of 37 questions: 20 directly related to AK sharing, and 17 related to the context in which the AK sharing took place.

7.3 Analysis

The analysis of the 97 valid survey responses was performed in three phases: first, the current state of AK practice and challenges was established by comparing the respondents' answers to the 20 AK related questions. The analysis of four of these questions is presented in §7.3.1: three questions about AK practices and one about challenges in AK sharing. In phase one, we examined the responses by ordering and grouping them.

Second, the relationship between the AK practices and challenges and their context was analyzed by determining significant correlations between the AK-related responses and some of the 17 context-related questions. The two context factors of project success and project size are analyzed systematically in §7.3.2. The result of phase two is a set

of statistically significant correlations between responses to AK related questions, and the size and success of the projects they pertained to.

In the third phase of the analysis, we reasoned and discussed about the results from the first two phases. Based on reasoning, literature and the experience of seasoned architects we deduced causality and impact mechanisms from the correlations, leading to an observed impact model that is presented in §7.3.3. Further discussions are presented in §7.4.

7.3.1 State of AK sharing practice

In this section, the responses to four of the AK related questions are analyzed, presenting the results of phase 1 of the analysis.

The four questions are:

- What type of architectural knowledge have you provided to or acquired from Logica in your latest assignment?
- Why did you share architectural knowledge to your colleagues in Logica?
- When did you share architectural knowledge in your latest assignment?
- What challenges in architectural knowledge sharing did you experience in your latest assignment?

Each question was provided with a set of predefined responses, determined in consultation between two experienced architects and two researchers. There was also the possibility for open text for missing answers. Respondents were asked to signify the applicability of those responses on a 5-point Likert scale. Table 7.1 lists the predefined responses to the questions, sorted by their average response values, which are listed in the third column. Each question is further analyzed in the following subsections. The two rightmost columns in the table list the Spearman's ρ correlations between the responses and the project context factors, which will be analyzed in §7.3.2 below. We will start with the analysis of the responses without taking into account their contexts.

Architectural knowledge types

What type of architectural knowledge have you provided to or acquired from Logica in your latest assignment?

Table 7.1: AK related responses, average values and correlations

Architectural knowledge types	**ID**	**avg**	**pr succ** ρ	**pr size** ρ
Standards; principles and guidelines	s_akt_std	2.95	-0.062	0.010
Tools and methods	s_akt_tlsmeth	2.80	-0.096	.213*
Known and proven practices	s_akt_prctc	2.71	0.135	-0.09
Product and vendor knowledge	s_akt_prodkn	2.71	0.187	-.212*
Requirements	s_akt_req	2.71	0.178	-0.079
Design Decisions including alternatives; assumptions; rationale	s_akt_dd	2.69	0.1	-0.011
Business knowledge	s_akt_buskn	2.61	0.082	-0.037
Patterns and tactics	s_akt_ptrn	2.46	0.044	0.023
Reference architectures	s_akt_ra	2.28	0.074	-0.025
Legal knowledge	s_akt_legal	1.79	0.097	0.117
AK Sharing Motivation	**ID**	**avg**	**pr succ** ρ	**pr size** ρ
To build up my professional network	s_akw_bldnetw	3.89	-0.116	0.087
I just like to share my knowledge	s_akw_like	3.84	0.115	-0.075
Personal relation with colleague(s)	s_akw_persrel	3.81	-.230*	0.127
We all work for the same company	s_akw_samecomp	3.77	0.109	-0.151
To enhance my professional reputation	s_akw_reput	3.59	0.042	0.009
To contribute to the company's business goals	s_akw_compbusgls	3.53	0.054	-0.002
I hope the favour will be returned some day	s_akw_return	3.39	-.204*	0.201
I will be recognised as a contributor	s_akw_recog	3.32	0.018	-0.079
I have received useful information from him/her	s_akw_reciproc	3.32	-.223*	0.020
My management expects me to	s_akw_mgtexpect	3.09	.275**	-0.103
This may work in my favour at my next salary review	s_akw_salary	2.69	0.002	-0.009
AK Sharing Timing	**ID**	**avg**	**pr succ** ρ	**pr size** ρ
Whenever needed to solve problems	s_akh_problems	3.48	0.153	-0.019
At the end of the project	s_akh_prjend	3.41	0.027	0.012
When colleagues ask me to do so	s_akh_collask	3.39	0.048	0.000
When management ask me to do so	s_akh_mgtask	2.59	0.177	-0.026
Whenever I have time	s_akh_freetime	2.57	-0.025	0.081
In the evening	s_akh_evening	2.53	0.012	-0.056
Continuously during the project	s_akh_prjcnt	2.34	.205*	-0.159
AK Sharing Challenges	**ID**	**avg**	**pr succ** ρ	**pr size** ρ
Difficulty to achieve common understanding of requirements	s_chl_requnders	3.82	-0.146	0.052
Difficulty to achieve appropriate participation from relevant stakeholders	s_chl_stkhpart	3.66	-0.165	0.036
Diversity in customer culture and business	s_chl_custdiv	3.61	-0.102	0.084
Poor quality of information	s_chl_infqual	3.42	-0.11	0.105
Lack of information	s_chl_inflack	3.31	-0.086	0.169
Inconsistency in information obtained from different sources	s_chl_infincons	3.26	-0.114	0.146
Lack of time	s_chl_time	3.25	0.06	-0.003
Delays in delivery	s_chl_delays	3.24	-0.167	0.152
Difficulty of obtaining the appropriate skills within the project	s_chl_skills	3.24	-0.115	0.138
Conflicts and differences of opinion	s_chl_conflict	3.19	-.214*	0.176
Difficulty to organise effective meetings	s_chl_effmeet	3.09	-0.153	0.211*
Lack of informal communication	s_chl_lackinformal	3.01	-0.204	.261*
Inaccessibility of technical facilities	s_chl_tinacc	2.99	-0.183	.280**
Growing and shrinking of project population	s_chl_growshrink	2.82	-0.117	.357**
Lack of trust between the project locations	s_chl_sitetrust	2.77	-.272**	.265*
Project personnel turnover	s_chl_persto	2.67	-0.116	.307**
No appreciation from (project or competence) management	s_chl_mgtappr	2.60	-0.125	.230*
No willingness to share knowledge	s_chl_nowill	2.39	-.224*	.245*

* Correlation is significant at the 0.05 level (2-tailed); ** Correlation is significant at the 0.01 level (2-tailed).

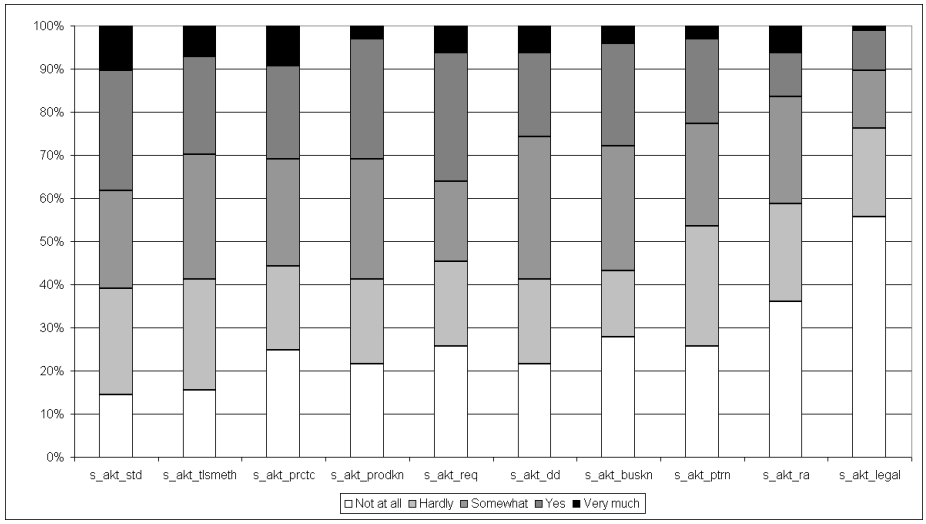


Figure 7.1: Architectural Knowledge Types

The distribution of the response values is visualized in Fig. 7.1.[1] With the exception of reference architectures and legal knowledge, all types of architectural knowledge appear to be shared more or less equally. The least shared type of AK is legal knowledge: over 75% indicate they do not or hardly share it with Logica.

AK sharing motivation

Why did you share architectural knowledge to your colleagues in Logica? The distribution of the response values is visualized in Fig. 7.2. These data tell us that most architects are either impartial to or agree with almost all motivation responses.

The only motivation that more architects disagree with (38%) than agree with (17%) is salary. A related finding is the unpopularity of management expectation as a motivator: 65% of respondents are impartial to or disagree with this motivator.

AK sharing timing

When did you share architectural knowledge in your latest assignment?

[1] The figures in this chapter use the codified response IDs of the *ID* column in Table 7.1.

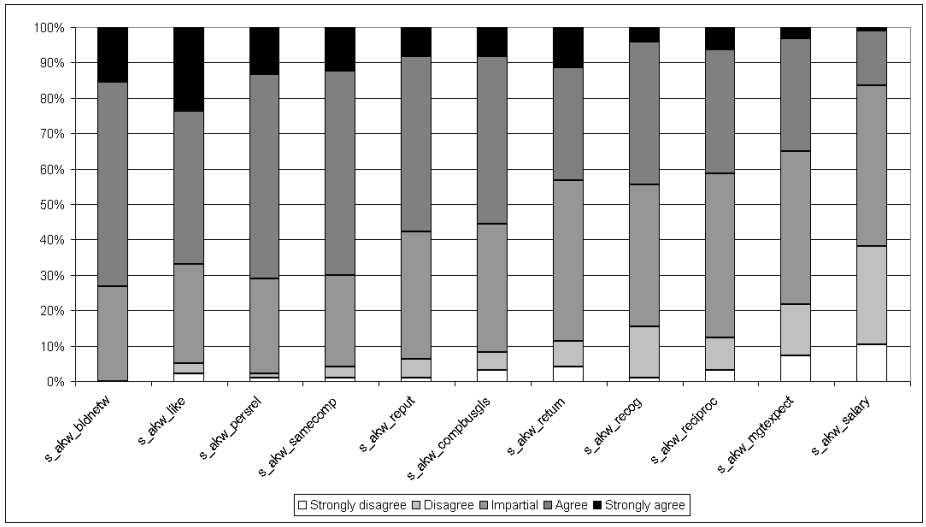


Figure 7.2: AK Sharing Motivation

The distribution of the response values is visualized in Fig. 7.3. By far the most popular times to share AK are when problems occur, at the end of projects and when asked by colleagues (other than managers); these three timings are all used often or very often by over 50% of the architects. Almost 30% of architects indicate they never share AK "when management asks me to do so". We assume this is because in those cases management does not ask - an assumption supported by the observation that there is no lack of willingness to share (see Fig. 7.4). This fortifies our previous observation about management expectation as a motivator.

AK sharing challenges

What challenges in architectural knowledge sharing did you experience in your latest assignment?

The distribution of the response values is visualized in Fig. 7.4. The ordering of the challenges by average response value in Table 7.1 allows an interesting categorization of challenges with descending response values:

- *Difficulty to achieve common understanding of requirements, participation from relevant stakeholders, and diversity in customer culture and business* (s_chl_requnders, s_chl_stkhpart, s_chl_custdiv) are all related to communication issues on

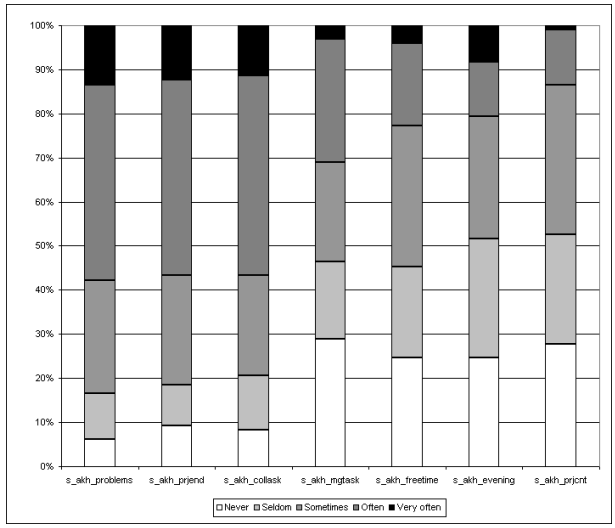


Figure 7.3: AK Sharing Timing

group level (as opposed to personal level); this is the category of challenges that most architects consider relevant in their latest projects.

- *Poor quality, inconsistency or lack of information* (s_chl_infqual, s_chl_inflack, s_chl_infincons) are about issues with quality or absence of codified AK; this is the second most commonly relevant category of challenges.

- *Lack of time and delays in delivery* (s_chl_time, s_chl_delays) are related to planning; this is the third most commonly relevant category of challenges.

- *Other challenges* all less commonly relevant than the three categories mentioned above, are related to obtaining resources, interpersonal issues, teaming, continuity and management.

In discussions about challenges in knowledge sharing, "knowledge is power" [Bacon, 1597] is often cited as a reason for professionals not to want to share knowledge. In our survey however, *lack of willingness to share knowledge* emerges as the least relevant challenge, which the majority of architects find irrelevant, and which only 18% find relevant. The next least relevant challenge is *lack of management appreciation*, which only 21% find relevant. The unpopularity of this response suggests that, even

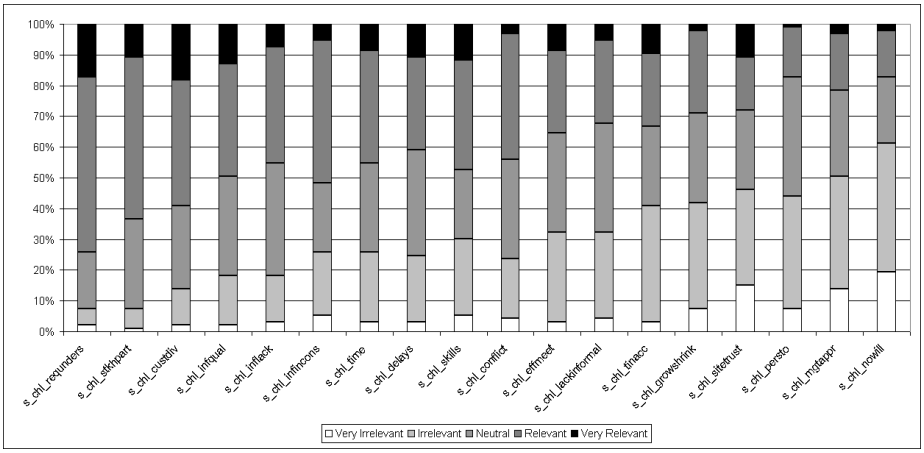


Figure 7.4: AK Sharing Challenges

though we have seen in §7.3.1 that both salary and management expectations are at the bottom of the list of reasons to share AK, architects are not actively discouraged by their management's apparent disinterest. Seeing that only 35% of respondents see management as a motivator (Fig. 7.2) and only 20% see management as a challenge (Fig. 7.4), one might conclude that *architects do not see management as an important factor in architectural knowledge sharing*. As we will see in the rest of this chapter, they might be wrong about this.

7.3.2 AK practices in context

In this section, we analyze the relationship between the AK practices and challenges and their project context, by examining significant correlations between the AK-related responses and some of the context-related questions. The two context factors analyzed here are project success and project size.

The first context factor analyzed is project success, as perceived by the architects. Perceived project success[2] is determined by asking the architects how they rated seven aspects of project success on a 5-point Likert scale from Poor to Excellent. The aspects they rated are: Sticking to budget, Delivery in time, Client satisfaction, Management support, Personnel turnover, Solution quality and Team satisfaction. The combined an-

[2]In this chapter, we use the terms "project success" and "perceived project success" interchangeably, always meaning the success as perceived by the architects and reported in the survey

swers of these seven aspects were subsequently averaged to obtain a quantification of overall project success per case. Cronbach's alpha test for internal consistency [Cronbach, 1951] was used to verify that these seven responses measure the same construct of success ($alpha = 0.82$).

Project size is determined by asking the architects for the number of project members.

Table 7.1 shows the Spearman's ρ correlations between project success and the AK practice related responses in column *pr succ* ρ. Correlations between project size and the AK practice related responses are in column *pr size* ρ.

Correlations with a positive or negative slope of over 0.2 and a significance level of under .05 (indicated by one or two asterisks) are considered significant and discussed here. In the discussion of the correlations, some speculation is presented as to the underlying mechanisms, based on our experience as practicing architects.

Cause and effect

One of the objectives of this survey was to gain insight into mechanisms around architectural knowledge sharing in projects. In other words, we were looking for ways in which architectural knowledge sharing impacts projects and vice versa - questions of cause and effect.

When analyzing correlations like the ones found in this survey, the question of causality between the correlated measurements deserves careful consideration. The mere presence of a correlation by itself does not imply a causal relationship. In order to determine potential causality, we resorted to three additional means: reasoning, literature and the experience of practicing architects in Logica.

The four categories of measurements we are correlating here are:

AKS Practices: the responses related to the type, motivation and timing of architectural knowledge sharing.

AKS Challenges: the responses to the question: *"What challenges in architectural knowledge sharing did you experience in your latest assignment?"*.

Project Success: the perceived success of the respondents' latest project.

Project Size: the size of the respondents' latest project (number of project members).

There are six possible correlations between these four categories. We are not analyzing correlations between AKS Practices and Challenges. Fig. 7.5 visualizes potential causality arrows for the five remaining possible correlations. In this figure and Fig. 7.8, a causality arrow from A to B symbolizes that A has impact on B, implying

that making changes to A would cause related changes in B. The arrows are based on the following reasoning:

Project Size ↔ Project Success Project size is well known to influence project success in many ways, both in literature [Frederick P. Brooks, 1995, Jones, 2000] and experience, so the primary arrow of causality is from Size to Success.

Project Size ↔ AKS Practices Experience indicates that mechanisms that determine project size are only marginally impacted by architectural knowledge sharing; on the other hand, project size determines factors like organizational and physical distance between project members, which are obvious factors in AKS. We conclude that any correlation found means that project size impacts AKS, and not the other way around.

Project Size ↔ AKS Challenges Like with AKS Practices, project size causes AKS challenges. There are some challenges that may in time conversely influence project size: for example, difficulty to obtain the appropriate skills may either lead to a smaller project because there is no staff available, or to a larger project because the lower skill level is compensated by adding more staff. We conclude that there is a primary causal arrow from project size to AKS challenges, and a potential secondary reverse arrow.

Project Success ↔ AKS Practices Examples of causality in both directions are experienced: e.g., a more successful project may lead to a better atmosphere causing more knowledge to be exchanged, or conversely more knowledge sharing may contribute to a more successful project. We conclude that we cannot a priori attach causality direction to correlations found between project success and AKS practices.

Project Success ↔ AKS Challenges The word *challenge* is used here as a synonym for *obstacle*, which can be defined as *something that makes achieving one's objectives more difficult*. Since the objective here is a successful project, the primary arrow of causality is by definition from Challenge to Success. There is also a possibility of reverse causality here: challenges may be exacerbated or caused by (lack of) project success, e.g. the atmosphere in an unsuccessful project may lead to lack of trust.

The causality arrows between the four categories of measurements as visualized in Fig. 7.5 will be elaborated in §7.3.3, based on correlations measured.

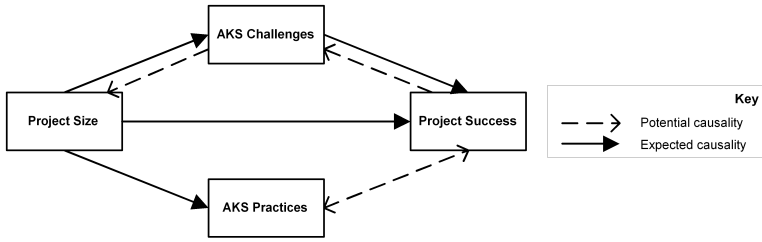


Figure 7.5: Causality as deduced from reasoning, literature and experience

Correlation with project success

We now discuss the correlations between architectural practices and challenges and project success. In column 4 of Table 7.1, we find 8 significant correlations. Summarizing:

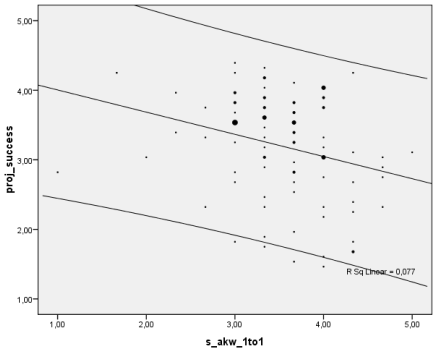
In *more successful* projects, architects tend to:

- be *less* motivated to share AK for interpersonal relationship reasons, but are more motivated by their management's expectations
- face *less* challenges related to interpersonal relationships.

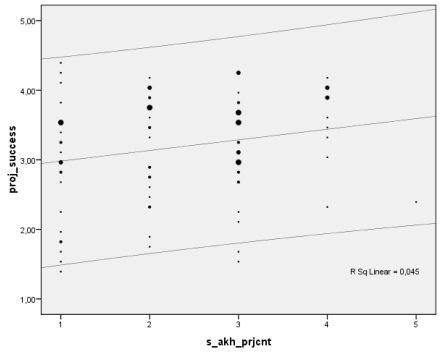
We find no correlation between project success and the type of the architectural knowledge shared.

- Motivation: *Personal relation with colleagues, or because I have received or hope to receive information from the other* (s_akw_persrel, s_akw_return, s_akw_reciproc): remarkably, all motivation responses that are related to one-to-one relationships between colleagues show a significant negative correlation with project success. Fig. 7.6(a) visualizes this relationship, showing a clearly downward slanting cluster: the x-axis represents the individual architects' average mark given to these three responses.[3] There are many possible explanations, but in view of our findings about AK sharing challenges a few items further down, the most plausible one appears to be related to trust. Problems in projects tend to reduce trust, which might cause architects to place more value on interpersonal motives.

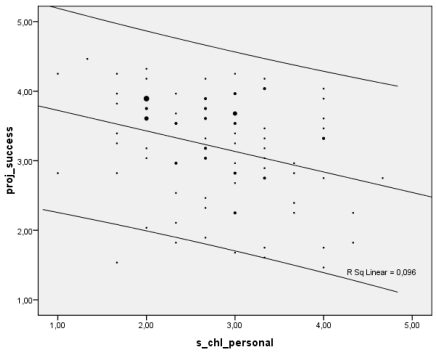
[3]The lines in the scatter plots in this section represent linear regression fit lines and their 95% confidence interval



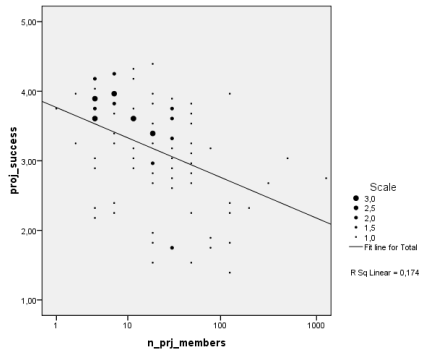
(a) Motivation: interpersonal relationships



(b) Continuous AKS



(c) Interpersonal challenges



(d) Project Size vs Success

Figure 7.6: Various AKS parameters plotted against project success

- Motivation: *My management expects me to* (s_akw_mgtexpect): even though management expectations are considered one of the least important motivations for sharing AK by the architects, it is the only motivation that has a positive correlation with project success. The explanation may also be related to trust levels: architects working on successful projects have more confidence in their management, and hence are more inspired or motivated by them.
- Timing: *Continuously during the project* (s_akh_prjcnt): the only AK sharing timing response that has a correlation with project success. However, visual inspection of Fig. 7.6(b) suggests that this is a spurious effect.
- Challenges: *Conflicts and differences of opinion, Lack of trust between the project locations,* and *No willingness to share knowledge* (s_chl_conflict, s_chl_sitetrust and s_chl_nowill). Since there is by definition a causality between AKS challenges and project success, we *expect* to find correlations. Remarkably, only three challenges are significantly correlated with project success. These three challenges, all with a very clear negative correlation, have in common that they are related to interpersonal relationships and emotion: conflicts, trust and willingness. We have plotted the correlation between project success and the individual architects' average mark given to these three responses related to interpersonal challenges in Fig. 7.6(c). As for the other challenges, finding *no* correlation indicates one of two things: either the challenge is so insignificant that the correlation is too small to be measured in a sample this size, or the challenge is somehow overcome or neutralized.

From these correlations, we can draw the following conclusion: the only significant AKS challenges that are not overcome or insignificant in projects, are those related to emotion and interpersonal relationships. In less successful projects, there is less trust and willingness to share AK, and more conflict. This appears be unrelated to the type of AK shared. There is, however, a significant correlation with architects' motivation to share architectural knowledge: in more successful projects, they are more motivated by management and less by interpersonal relationships between colleagues.

Correlation with project size

We proceed to discuss the correlations between architectural practices and challenges and project size, as documented in column 5 of Table 7.1. We find 10 significant correlations. Summarizing:

In *larger projects*, architects tend to:

- face significantly *more* challenges of multiple kinds

- share *more* knowledge about tools and methods, but *less* about products and vendors.

Project size has no effect on AK sharing motivation or timing.

- *Information related to tools and methods* (s_akt_tlsmeth) is shared slightly more by architects in larger projects than by architects in smaller projects. This is likely due to the fact that there are simply more developers to educate on tools and methods.

- *knowledge related to products and vendors* (s_akt_prodkn) architects in some smaller projects tend to share more. We suspect that this is due to the fact that in larger projects, decisions about products and vendors are often made on a higher (management) level, whereas smaller project architects are more likely to be involved in these decisions, and hence have to share more knowledge related to products and vendors.

- *AKS challenges* Table 7.1 shows that out of the 18 types of challenges surveyed, 8 are significantly correlated to project size. We have also calculated the aggregated AKS challenge level as the average of each architect's challenge-related responses. It turns out this aggregated AKS challenge level is correlated to project size with a correlation coefficient of 0.356 at a 0.001 significance level. The eight challenges at the bottom of Table 7.1 are the only ones that are also individually correlated to project size. Apparently, some challenges are universal, and others are considered less relevant in smaller projects, bringing down their average response value. We have illustrated this by plotting the average response values of both the seven least commonly relevant and the eleven most commonly relevant challenges against project size in Fig. 7.7. The figure confirms that there is indeed a clear upward trend, and that it is steeper for the less commonly relevant challenges.

  Based on the fact that larger projects are likely to include more distinct departments or locations, and the well-known issue of tension between departments, we would expect larger projects to suffer more from emotion-related challenges. We do indeed find correlations between project size and lack of both willingness (.245) and trust (.244), but no significant correlation with the challenge of conflicts and differences of opinion.

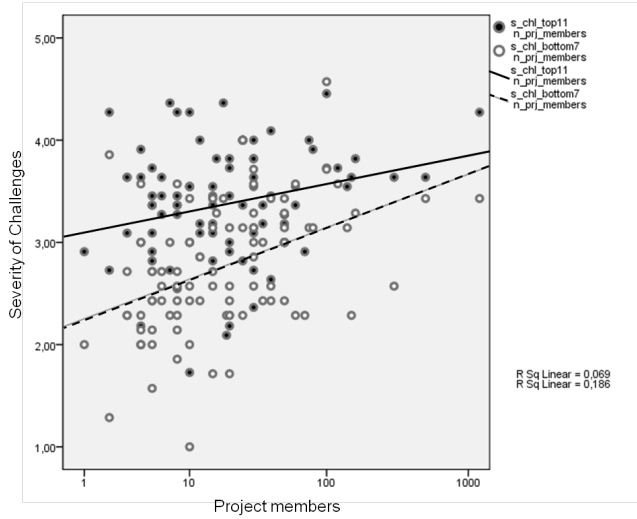


Figure 7.7: AKS Challenges versus project size

7.3.3 Refined model of causality

We now use the correlations observed in the previous section to obtain a more detailed picture of causality. Fig. 7.8 shows the causality arrows between the four categories of measurements as explained in §7.3.2 and visualized in Fig. 7.5, but the AKS category boxes have been replaced with more specific subcategories corresponding to the responses that showed correlations. Additional symbols show whether correlations are positive or negative. Specifically, we have:

- replaced the generic box *AKS Challenges* with a box *Less common AKS Challenges*, representing the seven least common AKS challenges that have significant positive correlations with project size
- created a box *Interpersonal challenges* inside the *Less common AKS Challenges* box, representing the three challenges related to willingness, trust and conflict that are negatively correlated with project success
- replaced the generic *AKS Practices* box with four specific boxes representing the practices that we have found to be correlated with either project size or project success

- added + and - symbols to the causality arrows representing the sign of the observed correlations.

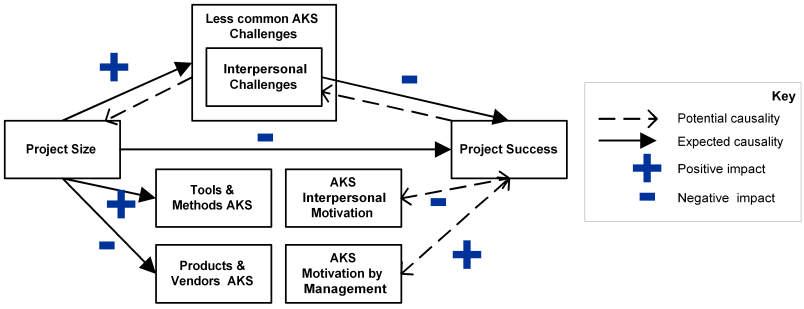


Figure 7.8: Causality as observed

There is one correlation that we had not discussed yet: that between project size and perceived project success. Fig. 7.6(d) displays a very clear correlation between project size and perceived project success. Perceived project success and project size show a negative Spearman's ρ correlation coefficient of -0.453, with a significance of 0.000. This is in line with results found by [Jones, 2000], and conversely provides some additional validation that our input data behave according to known properties of IT projects. [Frederick P. Brooks, 1995] gives a clear explanation of one of the mechanisms that cause this correlation. Surprisingly, a more recent survey [Emam and Koru, 2008] does not find this correlation.

Fig. 7.8 summarizes in one picture the combined mechanisms in the interplay between AKS and project size and success. We see how project size impacts some challenges, and which challenges impact project success. We also see that project size impacts the type of knowledge shared, and we observe a relationship between AKS motivation and project success, a relationship with an as yet undetermined arrow of causality.

7.4 Discussion and Related Work

In this section, we further discuss the results found above and threats to validity, and we relate them to additional related material found in literature. Please refer to §4.2.1 for a discussion of the project success construct and related work, which also applies to this chapter.

7.4.1 Threats to validity

These results are based on a survey of architects in one IT services company in one country. This limitation is somewhat softened by the fact that 64% of respondents work mostly at customers' sites, but the results are certainly influenced by cultural aspects of both the Logica company and the Netherlands location. It would be very interesting to repeat the survey in other companies and locations.

The ordering of the responses in Table 7.1 and the response value distribution bar charts is based on average response values. The meaning of the average number itself is not clear, since the Likert-scale is not equidistant. An alternative ordering quantity would be the percentile responses of e.g. the two most positive Likert values. This would have the advantage of being able to say exactly what the ordering quantity means, but the disadvantage of ignoring the information inherent in the detailed distribution of responses. Visual inspection of the bar charts shows that, with the exception of Fig. 7.1, the order of the responses would not be that much different, specifically in those cases where we have based reasoning on the response ordering. As an example: the "seven least commonly relevant challenges" in Fig. 7.4 that we have discussed above would also be the seven bottom-most challenges if ordered by percentile of respondents answering "Relevant" or "Very Relevant".

There is a weakness in the four questions analyzed in §7.3.1, in that they all appear to have slightly different scopes for AK sharing: two of the questions are about sharing towards or from Logica, one is explicitly about sharing with colleagues, and two are explicitly from the perspective of the originator. These scope differences are ignored in the analysis, since they cannot be remedied without redoing the survey.

A final threat is caused by our approach of doing multiple statistical tests, and deriving our model from significant statistical results found in those tests. This approach implies a risk of introducing spurious statistical results in the model. We have mitigated this risk by using reasoning, experience and literature, but it would be interesting to further validate the model by using it to predict results in other surveys.

7.4.2 Architectural knowledge sharing

Over the last years, much has been published on the topic of architectural knowledge sharing. The GRIFFIN project [Farenhorst and de Boer, 2009, Clerc, 2011, Ali Babar et al., 2009] and six SHARK workshops [SHARK, 2009] on SHAring and Reusing architectural Knowledge have been especially productive. [Farenhorst and de Boer, 2009] reports on challenges to sharing architectural knowledge: they examine these challenges in an IT company, but perform only a qualitative analysis. The authors deduce a number of issues resulting from a lack of architectural knowledge sharing,

but do not directly relate the challenges to project success.

7.4.3 Motivation and emotion

An interesting finding about motivation in this survey is the observed shift in motivation source from colleagues to management in more successful projects. Could there be an either/or effect, in the sense that the 1-on-1 motivation by colleagues and motivation by management are somehow mutually exclusive? In that case, one would expect a negative correlation between these two motivation sources, which we did not measure (Spearman's $\rho = 0.107$ with a two-tailed significance of 0.295). We conclude that the mechanisms causing these shifts are independent. The finding does, however, cause one to wonder about architects' apparent indifference to management expectations as either a motivator or a challenge. The well-known Chaos Reports [Standish Group, 1994] already showed empirical evidence for management attention being a key project success factor.

Markus already identified the importance of being aware of one's motivation long before the term *architect* was used in the context of system design: "Self-examination of interests, motives, payoffs, and power bases will lend much to the implementor's ability to understand other people's reactions to the systems the implementor is designing..." [Markus, 1983]. In literature, motivation is reported to have the single largest impact on developer productivity [Boehm, 1981, McConnell, 1996]. Moreover, in system development, the architecture represents the system's earliest design decisions with the highest impact on success [Bass et al., 2003]. Combining these facts, it is only to be expected that the motivation to share architectural knowledge is correlated with project success. Our results not only point to the importance of motivation and its source, but also shed some light on the mechanisms through which motivation and emotion impact project success through architectural knowledge management.

Finally, some words on the topic of *emotion*, a term that we introduced in §7.3.2 as the common element between the three only challenges that have a significant negative correlation with project success: *Conflicts and differences of opinion, Lack of trust between the project locations* and *No willingness to share knowledge*. During the analysis, we often wondered how it was possible that we did not find any significant correlation between the *other* challenges in AKS and Project Success. Consider, for example, the most commonly encountered challenge: *Difficulty to achieve common understanding of requirements*. How can a project be successful without common understanding of requirements? As stated above, the only plausible explanation is that all of these other challenges are apparently neutralized. With neutralize we mean that if these challenges occur, there are other factors that prevent them from having a significant impact on project success. In the case of our example, these could be compen-

sating activities to promote the common understanding of requirements, such as client meetings. In the end, the only challenges that are not neutralized are those related to lack of trust, willingness, conflicts and differences of opinion: all issues in interpersonal relationships that have a strong negative emotional connotation. Apparently, it is harder for architects to neutralize challenges when such negative emotions are involved. This is a phenomenon that practicing architects often observe in real life, and it should be no surprise, given that architects are human beings. The significant finding here is that these emotional challenges are not neutralized where all other challenges are, and hence they merit extra attention, leading to the warning in our title: *Beware of Emotions*.

We conclude:

FOR ARCHITECTS, TO UNDERSTAND THEIR MOTIVATION AND DEAL WITH EMOTIONS ARE CRUCIAL KNOWLEDGE SHARING SKILLS.

7.5 Conclusions

We set out on this survey with two goals, which were both achieved: to establish the current state of architectural knowledge sharing in Logica and its customers, and to gain insight into the mechanisms around architectural knowledge sharing in projects. In order to gain this insight, we looked at architects' responses to four questions about AK sharing, and the correlations between these responses and their latest projects' success and size, and we reasoned about impact mechanisms and causality.

The analysis revealed the following mechanisms:

- Architects face many challenges sharing architectural knowledge in projects;
- these challenges are more numerous and diverse in larger projects than in smaller ones.
- The most common of these challenges are related to group level communication issues, the quality of codified knowledge and planning issues;
- however, these common challenges are not correlated with project success, so apparently they are generally neutralized somehow.
- The only challenges that *are* correlated with project success are the ones related to interpersonal relationships: conflicts, trust and willingness to share knowledge.
- Architects' motivation to share knowledge is more personal in less successful projects.

- Architects do not see management as an important factor in architectural knowledge sharing, but those architects that are motivated by management tend to work in more successful projects.

Our final conclusion is that *dealing with emotions* is a crucial factor in how architectural knowledge sharing leads to successful projects. It is important for architects to understand their motivation, and they should carefully deal with emotions when sharing knowledge.

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. We examine the consequences of this point of view on the architecture process, and give some guidance on its implementation, using examples from practicing architects trained in this approach.

8.1 Introduction

As mentioned in the introduction to this thesis, the notion of "software architecture" is one of the key technical advances in the field of software engineering 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, but practicing architects' decisions appear to have a much wider range.

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 recent years, this view of architecture and architecting has been especially beneficial in promoting insight into architectural knowledge management: among other results, it has led to new insights into the structure of what architects need to know and document to make their decisions [Tyree and Akerman, 2005, Ivanović and America, 2010a], and it has stimulated research into re-usable architectural decisions [SHARK, 2009, Zimmermann et al., 2007]. Both these avenues of research have devoted much attention to structure and tooling, but so far there is limited focus on what the architect's decisions should be *about* beyond components and connectors.

We propose to view architecting as a risk- and cost management discipline. In other words, the important things architects should make decisions *about* are those concerns that have the highest impact in terms of risk and cost. Of course, risk and cost have always been important drivers in architecture [Kazman et al., 2002], but our point of view goes a step further than this obvious point: we see risk- and cost management as the *primary business goal* of architecting. All architecture activities such as architecture documentation, evaluation and decision making are in essence ways to fulfill this business goal.

As we will see in this chapter, considering architecture as a risk- and cost management discipline, rather than merely as a high-level design discipline, makes the architect more effective in a number of ways:

- It helps the architect order their work, in both focus and timing of their decision making.
- It helps in communicating about the architecture with stakeholders in business terms.

The rest of this chapter is structured as follows:

- In §8.2, we will show how we arrived at this point of view of what architecture is all about, and define some key concepts.
- In §8.3, we will elaborate on the meaning of risk and cost and how they determine architectural significance.

- Then in §8.4, we will examine how viewing architecture as a risk- and cost management discipline impacts the architecting process and helps architects in the focus and timing of their decision making.
- In §8.5, we will use three examples from industry to illustrate how this approach helps to communicate architectural significance to stakeholders.
- In §8.6, we will give some guidance on how to implement this point of view.
- We conclude the chapter with a discussion, including some frequently asked questions about this approach.

8.2 What are Architectural Decisions About?

For years, we have been talking and writing about *software* architecture as a set of *design* decisions [Kruchten, 1998, Jansen and Bosch, 2005, Tyree and Akerman, 2005]. Hence, the topic of an architect's decisions is supposedly *software design*. If we take the slightly more inclusive accepted view of software architecture as the architecture of software-intensive systems [ISO 42010, 2011], this view of the world sees a software architect as someone who makes design decisions about software-intensive systems. On further scrutiny, this qualification appears to be too generic: not all design decisions about software-intensive systems can be called architectural. For example, programmers make minor design decisions whenever they are writing code, which are manifestly not architectural: if they were, every programmer would be called an architect.

An often heard distinction is that architects operate at a *higher level of abstraction* than designers or programmers [Shaw, 1990]. This certainly appears to be true of architects who operate mainly by establishing design principles. Many architects, however, do much more than that, all the way down to prescribing details of particular coding practices that they deem architecturally significant [Fowler, 2003].

Let's see if the international standard definition of software architecture [ISO 42010, 2011] helps: "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution". Going back to the example of our programmer and his daily minor design decisions, these can certainly be about the properties of the system, its elements and their relationships. The programmer's decisions may also affect the system's evolution. There are only two concepts in the ISO 42010 definition that are clearly out of the league of the programmer's decisions: "fundamental" and "principles". These concepts can guide us towards the class of decisions architects should focus on: decisions about fundamental

things and principles. Ralph Johnson, quoted by Martin Fowler [Fowler, 2003], makes a similar statement in less formal words: "Architecture is about the important stuff. Whatever that is."

In the last few years, the authors have started to equate "the important stuff" with "the stuff that has the most impact on risk and cost". In other words, architects focus on concerns that involve high risk and cost, and architectural decisions are those decisions that have significant impact on risk and cost. This view is a joint understanding that has come into being after five years of seeing IT architects at work on dozens of diverse complex solutions, and after extensive discussions with the architects and their stakeholders on what should be the focus of an architect's work. It has so far been beneficial in several ways:

First, it helps architects organize their workflow by giving them a relatively objective way of prioritizing concerns and determining when to make decisions. How it does this is explained in §8.4.2 and §8.4.3 below.

Furthermore, it helps architects discuss architectural significance with business stakeholders in terms that they all understand: risk and cost. This is explained in §8.5.

8.2.1 Key concepts

"Concern" and "Decision" are key concepts throughout this chapter. Concern is a well-understood, established concept: [ISO 42010, 2011] uses the term concern to mean *any topic of interest pertaining to the system*. Concerns originate from stakeholders' needs: this makes them "of interest" in the ISO42010 definition. A decision is a choice by the architect amongst alternatives. The architect makes decisions to Address (fulfill, satisfice, handle) concerns. The SEI's Attribute-Driven Design [Bass et al., 2003] is an example of decision-making where you choose from tactics to address a quality attribute concern. Some examples of architectural concerns from our experience:

- How to fulfill the requirement to instantly revoke a user's authorization, even in the middle of a session?
- How to implement required UI elements that are not supported by HTML?
- How to fulfill the response time criteria?
- When to upgrade to the new version of the application server platform?
- Which workflow engine to use?
- Whether or not to virtualize our server landscape?

As we can see from these examples, concerns can usually be phrased as a question, and it is the architect's task to decide on the answer to that question, thereby addressing the concern. Sometimes the concern is an open question, e.g.: "How to fulfill the

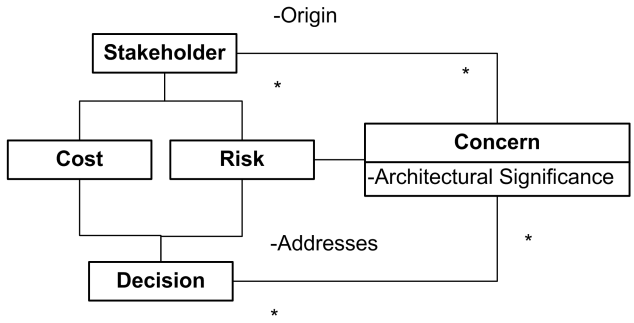


Figure 8.1: Key concept model.

client's security requirement?". Other times the concern is a closed question, e.g.: "Should we buy component A or build it ourselves?" or "Should we use thin or fat client technology?". The closed questions are usually elaborations of previous open questions, narrowing down the answer space after some research. We can usually see the decisions taking shape in these closed-question concerns: they point to a choice to be made between a number of alternatives. The word "concern" also has a connotation of "worry", and this is no coincidence: architects worry about fulfilling the concerns. They would like to prioritize the concerns that they worry most about, and consider those the most significant concerns at any point in time.

Several models exist linking design decisions to concerns [de Boer et al., 2007, Jansen and Bosch, 2005, ISO 42010, 2011]. [de Boer et al., 2007, Jansen and Bosch, 2005] represent the Concern-Decision relationship as a 1-to-n association that crosses other entities like "alternative" or "solutions", even though in our experience, architectural decisions regularly address more than one concern at the same time. For our purposes, these other entities are not needed, so like [ISO 42010, 2011] we simplify and generalize them to Fig. 8.1.

In our model, we see Stakeholders in three important relationships: as the origin of concerns, as the party that bears the cost of implementing decisions, and as the victim bearing the risk of wrong decisions. Please note that these are independent associations: the stakeholder paying for the implementation or suffering the risk of a wrong decision is not necessarily the same Stakeholder from whom originates the concern that the decision is addressing.

The final concept we would like to define here is Architectural Significance. This is an attribute of Concern. It is the key concept used in this chapter to describe the

amount of attention the architect should pay to a particular concern.

8.3 Architectural Significance in Terms of Risk and Cost

In §8.2, we stated that architects focus on concerns that involve high risk and cost. In this section, we make this statement more exact: *the architectural significance of concerns can be represented by their cost and risk level.*

Below, we present an approach for roughly quantifying the architectural significance of a concern. The purpose of quantifying the architectural significance of concerns is to be able to order them, so that architects can direct their attention to the most significant concerns as explained in §8.4. It should be noted that architects in industry mostly follow their gut-feeling and experience, supported by checklists and templates, for assessing what is architecturally significant, and that in our experience this assessment is usually strongly related to risk and cost. The formulas presented below are primarily a conceptual tool, a vehicle to explore the relationships between architectural concerns and decisions, and their architectural significance in terms of cost and risk. The usage of the formulas in practice is limited to those occasions where an architect feels the need to validate their gut feeling of architectural significance by a rough quantification, or needs to justify their prioritization towards business stakeholders in broad economic terms.

The principle on which our approach is based is this: *the architectural significance of a concern can be represented by the budget an organization would have to reserve to address that concern, including cost contingency,* or:

$$AS(C) = Cost(C) + Risk(C)$$

In this formula, $AS(C)$ represents the quantified architectural significance of concern C. $Cost(C)$ is the estimated cost of addressing the concern, as explained in §8.3.2. $Risk(C)$ is the total expected cost of "things going wrong" associated with C as explained in §8.3.1: the cost contingency.

An example of how this principle would be used to assess the architectural significance of a performance concern C_{perf}: $Cost(C_{perf})$ would represent the cost of addressing the concern, including the cost of designing and engineering work specifically aimed at achieving the required performance, performance testing and tuning, and cost of any tools used in this work. $Risk(C_{perf})$ would then be the cost contingency associated with the risk of not properly addressing the concern - this term will be elaborated in §8.3.1.

The formula represents architectural significance in terms of money. Apart from giving us a way to order concerns, it also gives us an idea of the maximum economic benefit that can be achieved by architectural activities related to these concerns. If the cost of these activities grows beyond this maximum benefit, it clearly makes no sense to continue them: they are not architecturally significant. A special case of this, regarding the cost of quantification of quality attributes, is presented by [Glinz, 2008]. Spending more resources on addressing concerns than is warranted by their impact in terms of risk and cost is a waste. Such concerns are clearly not architectural. Using risk level to determine what an architect should do, and especially *not do*, is the basis of Fairbanks's risk-driven model [Fairbanks, 2010]; we add cost as an equally prominent factor to consider.

8.3.1 Risk

A Risk is something that may go wrong. Traditional architecting activities control risk in a number of ways, both before and after committing to an architectural decision:

1. *Gathering information* (before committing), reducing the uncertainty of a wrong architectural decision by e.g. architectural prototyping or architectural analysis.
2. *Risk-mitigating design*, using architectural strategies and tactics that reduce the impact of change after committing, such as loose coupling and abstraction layering.
3. *Documenting architectures*, reducing the probability of misunderstanding architectural requirements and/or decisions.
4. *Evaluating architectures*, reducing the probability of wrong architectural decisions by having the architecture reviewed against critical criteria before committing.

The quantified definition of risk in this chapter is the risk exposure relationship used by Barry Boehm [Boehm, 1991]:

$$Risk(F) = p(F) \times I(F)$$

in which F is a particular failure scenario, $p(F)$ is the perceived probability of F occurring, and $I(F)$ is the estimated impact of F. When thinking about risks, normally cost is not the only impact if things go wrong: other impacts should not be forgotten, such as the impact of failure on delivery time or stakeholder satisfaction. When calculating risk exposure, all impact needs to be somehow converted to financial impact.

The reason we are talking about *perceived* probability and *estimated* impact is because the architect, at the time of architecting the solution, can never determine the actual probability and impact of failure.

We state that risk is an important factor in determining which concerns an architect should focus on. To be able to do this and quantify the risk aspect of a concern, the architect must tie architectural concerns to failure scenarios. Most concerns have inherent failure scenarios; this is what architects worry about. The most generic failure scenario related to a concern is "not addressing the concern", a failure to meet the stakeholders' needs underlying the concern. Sometimes obligation to meet the concern is formalized in an agreement (e.g. a Service Level Agreement), with related specified penalties: in those cases the direct financial impact to the supplier is equal to those penalties (but there may also be indirect impact, like reputation damage). Sometimes the impact is harder to express financially, like loss of life when not meeting a safety concern.

Once we have identified all independent failure scenarios related to a concern and estimated their associated probability and impact, we can determine the concern's financial risk impact by simply adding the financial risks of these scenarios. This calculation is exactly the same as calculating a project's required contingency budget based on its risk register [AACE, 2000], which we do not have to explain here.

An important aspect the solution architect should take into account is that stakeholders have different interests in risks, related to the difference of the scope of their stake in the solution. Stakeholders each have their own interests: a project manager will be mostly interested in the risks that impact project success, while a security officer will be more interested in risks that impact security, which usually occur only after the project has delivered the solution and the project manager has been discharged. This difference could be made visible in the model described above by making the impact estimate of failure scenarios stakeholder-dependent. One can then choose to add the risk exposure of all stakeholders, or to filter out the impact related to less critical stakeholders. What this tells us, is that the *architectural significance* of a concern is stakeholder-dependent, and architects have to be able to deal with diverging stakeholder views on which concerns are more critical to deal with. Many architects working in a project context feel an inherent responsibility for the lifetime of the system, extending beyond the project's end. The approach presented here gives them a way to quantify differences in stakeholder interests.

Risk example

Returning to our example performance concern C_{perf} above: failure scenarios are those turns of events in which the solution fails to meet the performance criteria, e.g.

the response times of a web application are too slow. Three examples failure scenarios for the web application that is too slow:

F_1 The hosting platform is underdimensioned.

F_2 The connection between the hosting platform and the internet is too slow.

F_3 The number of web-users exceeds expectations.

The architect has to assess these risks at design time – more precisely, at the time she is ordering the concerns to be addressed. F_1 and F_2 are design or construction "errors": the architect assesses the probability $p(F_1)$ and $p(F_2)$ of such errors based on her experience with similar solutions, probably taking into account the available budget, technical parameters and uncertainty. The impact of such construction errors $I(F_1)$ and $I(F_2)$ consists of the damage and repair costs. Damage is caused by e.g. contract penalties in the Service Level Agreement and/or users turning away from the system because of its unresponsiveness. Repair costs are the costs of adding hardware and/or bandwidth.

The architect assesses the probability of the number of web-users exceeding expectations $p(F_3)$ based on available knowledge of the uncertainty in the projected numbers of users. The impact $I(F_3)$ of this scenario again consists of damage and "repair" costs: "repair" perhaps being a strange term here, because the solution itself is not broken: it is just being overused, and needs to be expanded.

The total risk exposure then consists of adding the individual scenario's exposures: $p(F_1) \times I(F_1) + p(F_2) \times I(F_2) + p(F_3) \times I(F_3)$.

Once again, these calculations are not usually made in practice to any level of detail: they should be considered a conceptual tool, and applied in a manner commensurate with the objective of determining architectural significance. The architect usually considers only one or two failure scenarios that dominate the concern, and prioritizes based on rough order-of-magnitude impact assessments.

Decision risk

Apart from the risk of not addressing concerns, architects worry about another type of risk: the risk of making wrong architectural decisions. When generating a concern's failure scenarios, we should include scenarios in which the decisions addressing the concern turn out to be wrong. The two types of risk are closely related, and one could argue that a failure to address a concern is simply a failure to make the right decisions to address it. In [Poort and van Vliet, 2011] we presented a simple formula based on this argument, but we will delve a bit deeper here.

What is a "wrong decision"? It is when we choose A when B would have been better, or vice versa. "Better" meaning e.g.: costing less in terms of time and money, or resulting in a solution that better fulfills requirements. A failure scenario related to an architectural decision, expressed in its most generic terms, is: *it turns out we made the wrong decision*. We denote this failure scenario F related to decision D by $F_{D \rightarrow wrong}$. It is important to understand that "wrong" is not an attribute of a decision; "wrong decision" is simply a shorthand way of expressing a scenario. In such a scenario, the failure can be due to all kinds of internal and external factors, unforeseen events etc., which "it went wrong" pertains to. Hence, the word "wrong" does not necessarily qualify the architect's work when making the decision.

Let's look at an example concern from the software architecture world: keeping java objects in an application server in sync with the corresponding records in a relational database (RDB). This is known as the O/R mapping problem, and several tools and techniques exist to address it: use of the Hibernate tool, use of entity beans, hard-coded SQL, etc. We will base a small thought experiment on this concern. For simplicity's sake, let's state that the O/R mapping problem can be addressed by one architectural decision: the decision to choose one of the available tools or techniques. The failure scenarios are those that inhibit the solution's quality attributes like maintainability, data integrity, performance etc. Depending on the context and what happens in the future, a decision to hard-code SQL statements to populate java objects *could* turn out to be wrong. The resultant close coupling of the java code with the RDB data model and technology *could* cause excessive effort to be needed for changes, violating a modifiability requirement. We cannot know this at design time; all we can do is assess the probability $p(F_{D \rightarrow wrong})$ and the impact $I(F_{D \rightarrow wrong})$ of this failure scenario. This is not trivial, since the impact depends on when the failure is determined. If the situation goes undetected it will harm the stakeholders by driving up the cost of changes. If it is detected before the "wrong" solution can do any damage, the project team would still have to re-factor the code, and the impact of the wrong decision ultimately translates to the total cost of this re-factoring to the stakeholders, including the re-factoring effort and any additional costs caused by the subsequent delay in delivery of the solution. This is in effect the cost of *reversing* the architectural decision.

This thought experiment illustrates a general point: the impact of a wrong decision usually involves both the cost of reversing the decision and the potential damage to the stakeholders if it is *not* reversed, or *until* it is reversed. In any case, the cost of reversing an architectural decision is an important factor in its architectural significance. When using cost and risk to determine architectural significance, design decisions that are expensive to reverse tend to be more architectural. This gives a theoretical basis to Fowler's qualification of architecture as "things that people perceive as hard to change." [Fowler, 2003]. It also resonates strongly with [Klusener et al., 2005], who

state that "the software architecture of deployed software is determined by those aspects that are the hardest to change."

8.3.2 Cost

Cost is the amount of money spent on something. The formula for estimating the cost of addressing a concern is:

$$Cost(C) = \sum_{D \epsilon DA(C)} Cost(D)$$

where $DA(C)$ is the set of decisions addressing concern C, and $Cost(D)$ is the estimated cost of implementing decision D. There are many documented ways to estimate cost in software engineering (e.g. [Boehm, 1981]), which we will not discuss here. Risk factors must be excluded from this cost estimate, to prevent double-counting risks into the architectural significance function $AS(C)$ above. Once again, we are not suggesting to use this formula to determine architectural significance in practice; it is presented to clarify our view on architectural significance.

It should be noted that concerns and design decisions have an n-to-m relationship: design decisions often address multiple concerns, so that adding costs calculated this way for multiple concerns C_1 and C_2 will cause double-counting for decisions that address both concerns in $DA(C_1) \cap DA(C_2)$. Since we are only interested in determining the cost-factor in architectural significance, we are not planning to add costs of different concerns, so this is no problem here.

Just like with risk, the solution architect should always realize that stakeholders have different interests in costs, related to the difference of the scope of their stake in the solution. Hence, a project manager will be mostly interested in project costs, while a business owner may be more interested in the Total Cost of Ownership (TCO). Depending on the solution architect's context, her architectural decisions may effect TCO, project costs or both. So both the cost and the risk element of architectural significance are shown to be stakeholder-dependent.

8.4 Impact on Architecting Process

We will now examine the impact of the risk- and cost management view of architecture on the architecting process. In the previous sections, we have discussed the importance of managing risk and cost in architecture, and presented a method for ordering concerns by architectural significance. In this section, we will show how this ordering can be

used to optimize an architecting process so that it becomes better at controlling risk and cost.

As a reference architecting process, we once again use the generic approach documented in [Hofmeister et al., 2007]. We have already encountered this paper in Chapter 6: it compares five industrial approaches to architectural design, and extracts from their commonalities a general software architecture design approach.

The approach involves three activities, and a workflow that reflects the fact that the three activities are not executed sequentially.

8.4.1 Architecting activities

The generalized architecting activities are:

1. *Architectural analysis*: define the problems the architecture must solve. This activity examines architectural concerns and context in order to come up with a set of Architecturally Significant Requirements (ASRs).

2. *Architectural synthesis*: the core of architecture design. This activity proposes architecture solutions to a set of ASRs, thus it moves from the problem to the solution space.

3. *Architectural evaluation*: ensures that the architectural design decisions made are adequate. The candidate architectural solutions are measured against the ASRs.

Of these three activities, the one that is most impacted by the risk- and cost management view of architecture is *architectural analysis.* The basis of this analysis are the solution's context and architectural concerns. Viewing architecting as a risk- and cost management discipline sheds light on which concerns are architectural: those that have high impact in terms of risk and cost. This addition necessarily implies that risk and cost assessment becomes part of the architectural analysis activity. This applies not only to the concerns to be addressed, since the impact in terms of risk and cost is transferred to the Architecturally Significant Requirements (ASRs) resulting from the analysis. In [Hofmeister et al., 2007], the ASR definition is borrowed from [Obbink et al., 2002]: "a requirement on a software system which influences its architecture". In risk- and cost driven architecting, the ASRs are likely the most sensitive requirements in terms of risk and cost.

8.4.2 Architecting workflow

In [Hofmeister et al., 2007], the authors describe an "apparently haphazard process" in which architects maintain, implicitly or explicitly, a "backlog of smaller needs, issues,

problems they need to tackle and ideas they may want to use. The backlog drives the workflow, helping the architect determine what to do next." Hofmeister et al. make clear that the backlog typically consists of architectural concerns but also other types of items, and that it is constantly re-prioritized using various prioritization tactics. They mention risk as one of the mostly external forces driving priority; others are team or stakeholder pressure or perception of greater difficulty.

When talking to architects, many of them indicate that this is where most of the added value of the risk- and cost driven view on architecting is. It helps them better organize this apparently haphazard backlog process by giving them a clear measure of priority: prioritizing by risk and cost.

In §8.3, we have only discussed determining the architectural significance of Concerns. Backlog items typically take the form "We need to make a decision about X." or "We should look at Y in order to address Z."[Hofmeister et al., 2007]. The fact that not all of the backlog items are formally architectural concerns is not a problem in practice, as long as the team is not too religious about the definitions. As long as the backlog items can reasonably be expressed in terms of risk and cost, the prioritization works.

One important thing to realize here is that *A has higher priority than B* does not necessarily imply *A must be addressed before B*. The backlog is not a strict picking order. Rather, it helps to identify the top n items to be addressed at a particular point in time, where usually $n = 5 \pm 2$. This seems to be a rather unsophisticated approach, but as we will see in the next section, the risk management aspect of architectural decision making allows us to further analyze the timing aspect.

We saw in the previous section that the architecting activity that is most directly impacted by the risk- and cost management view of architecture is architectural analysis. The architecting workflow, however, drives *all* architecting activities, including some that are not mentioned in Hofmeister et al.'s generalized approach, such as architecture implementation and maintenance. Through the architecting workflow, the risk- and cost management view of architecture permeates into those activities as well. For example: in the architecture documentation activity, the views that should be documented first are those that are associated with concerns that have high impact in terms of risk and cost.

8.4.3 Architectural decisions and the flow of time

When discussing risk and time, an important aspect is that the nature of the risk of a wrong decision D changes at the moment that we commit to the decision. Until that moment, the primary failure scenario is "the architect *will make* a wrong decision"; after that moment, the failure scenario changes to "the architect *has made* a wrong decision". The difference seems trivial, but is not, as we will see in this section.

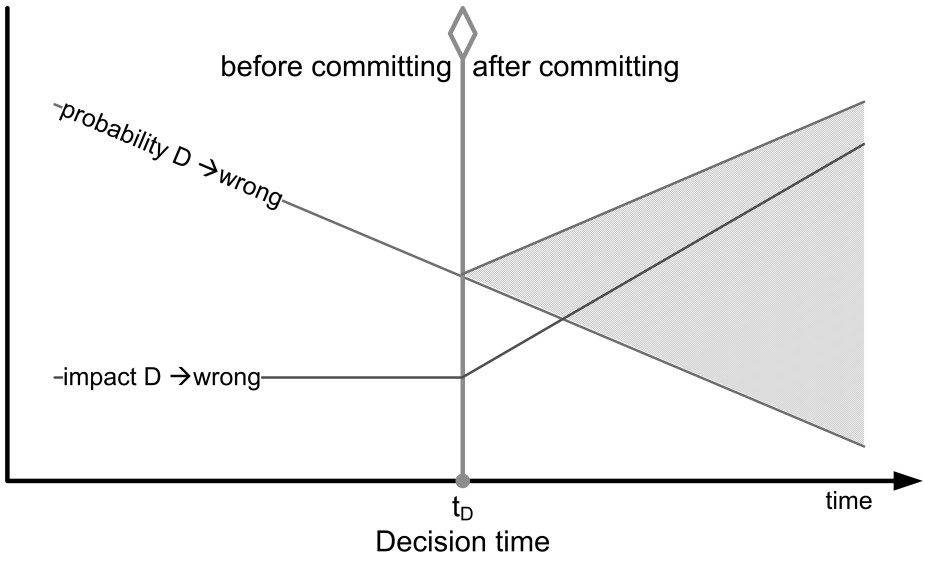


Figure 8.2: Decision risk factors over time.

The time at which to address architectural concerns is influenced by their risk-related character. We have seen in §8.3.1 that the risk related to an architectural decision is the product of the probability of a wrong decision and the impact of a wrong decision, and that an important component of the second factor is often the cost of reversing the decision. Generally, the influence of time on these factors depends on the moment of committing to a decision, as is illustrated in Fig. 8.2. In this figure, we see the probability that a decision D turns out wrong represented by a line, the impact of D being wrong by another line, and the moment t_D of committing to decision D as a vertical line:

- *After* the moment of decision, the cost of reversing an architectural decision, and with it the impact of it being wrong, will *increase* over time as it is being implemented.
- *Until* the moment of decision, the probability of a wrong architectural decision *decreases* over time as more information becomes available.
- Once we start implementing the decision, still more information will become available, but because we are already committed to it, it will not necessarily

reduce the probability of D *having been* wrong. This is represented by the shaded triangular area.

The shaded triangle requires some explanation. It is caused by the emerging information during the implementation of decision D, which can be either good or bad news:

- If all goes well, the probability of the decision having been wrong $P(D \rightarrow wrong)$ decreases, and the probability line will continue to go down. But whatever happens, the impact of D having been wrong will increase. The resulting risk may still either go up or down, depending on the rate of increase of the impact. $R(D) = P(D \rightarrow wrong) \times I(D \rightarrow wrong)$ may either decrease or increase.

- If the emerging information points more and more in the direction of D having been wrong, both $P(D \rightarrow wrong)$ and $I(D \rightarrow wrong)$ will increase, so $R(D)$ will certainly increase.

Assuming that the cost estimate of implementing D remains stable, this implies that the architectural significance of the concern C that D is designed to address may increase *after* we have made the decision.

Looking at architecting as a risk- and cost management discipline, we have to conclude that it is important for architects to continue to pay attention to the concerns they have made decisions about, because their architectural significance may yet increase. An extreme example of this is that sometimes concerns that did not seem architectural at the time of architecting, in running systems turn out to be architectural after all. So, risk- and cost driven architecting leads us to extend the list of architectural activities designed to control risks at the beginning of §8.3.1 with another activity: *monitoring architectural concerns after committing to decisions to address them.* The economic impact of monitoring decisions and resolving uncertainty over time has been analyzed extensively by several authors [Biffl et al., 2006] using decision trees and real-option theory.

8.5 Stakeholder Communication

Many stakeholders, especially business managers, are not used to thinking and talking in terms of levels of abstraction or components and connectors. Part of the architect's job is to translate architectural concerns and decisions into terms that stakeholders understand [Clements et al., 2007]. One substantial advantage of expressing architectural

significance in terms of risk and cost is that they are universal terms that most stakeholders can relate to. These terms smooth communication between architect and stakeholders, and gives the architect a relatively objective measure to explain priorities to stakeholders. We already saw the importance of objectivity in stakeholder communication about solution architecture in the SMSC case study in Chapter 5. In §8.5.1, we will list some more examples from practice of architectural concerns and how they can be expressed in terms of risk and cost to facilitate stakeholder communication.

Apart from the level of individual concerns, we are also experiencing that viewing architecting as a risk- and cost management discipline is improving business managers' understanding of the overall value of architecture and architects. Managers routinely understand the value of risk mitigation and cost control. When these are presented as the primary business goals of architecture, we find that this makes business managers more comfortable assigning often highly-paid [Money Magazine, 2010] architects to projects or product organizations.

8.5.1 Examples from practicing architects

In this section, we will present some examples from real projects[1], presented to us by the architects trained in the Risk- and Cost Driven Architecture approach. These examples highlight the risk and cost aspects of typical real-life architectural concerns:

- *Application Server platform* A large, business critical product application with a substantial java-based web interface is extensively customized and parameterized before being put in production. The development team is using the Open Source JBoss application server platform to develop the customizations. The target production platform is a Commercial Off-The-Shelf (COTS) application server. At a certain point in time, the development platform will also have to start using the target COTS application server. The architectural concern is the timing of this move. Moving the development platform to the "heavier" COTS application server too early will cause loss of efficiency and entail extra costs in the development team. Moving too late carries the risk of finding application-server specific issues too late, maybe forcing some refactoring that could have been avoided. The primary business stakeholder initially was not interested in this concern: according to his knowledge, J2EE application servers were standard and the move should be trivial. The architect had the important challenge of making the business stakeholders aware of the risk and cost aspects of the concern.

[1] All examples are from real projects; due to company confidentiality constraints, the examples have been abstracted away from their specific project context.

- *Role-based Interface* A web site has been designed and presented to the business stakeholders. In the design, the user only sees functionality that she is authorized for. Architectural concern is that users can switch roles during a session, which is not supported by the COTS portal platform in use (roles are cached during the session). Time to solve this issue is very limited. Several alternative solutions are considered: asking the portal supplier for help is risky, because it will take a long time and there is no guarantee for success. Alternatively, users can be required to log out and back in when they switch roles; this is low-risk, but makes the system less efficient, raising costs on the user side. In this example, making the risks and costs explicit helps the stakeholder make the right trade-off.

- *Web and SOA access channels* A large java-based application is being developed. The system will have a broker-like role, connecting various small and large companies, at widely varying levels of IT sophistication. The system is required to offer much of its functionality both as a web-based user interface (for small, low-IT companies) and as SOAP web-services (for larger companies with more sophisticated IT). At the time of designing the system, it is unclear what the distribution across these access channels will be in the near future; it might even change substantially during the time the system is being built. Key architectural decisions to address this concern are whether or not to build a common abstraction layer for both the web- and web-services interfaces on top of the business layer, and what mechanism to use to expose the common functionality to web-services. Costs are the development cost of the abstraction layer, the license and configuration costs of COTS web service integration packages. The key risks are jeopardizing performance by the abstraction layer, putting a lot of effort in access channels that might hardly be used by the time of deployment, and inconsistencies in business rules across the access channels.

8.6 Implementing the Risk- and Cost Driven View of Architecting

After elaborating the theoretical implications of viewing architecting as a risk- and cost management discipline in the previous sections, we will now focus on the practical implications for the architect's activities. We will do this in the form of a list of guiding principles that the architect can apply to their way of working. This guidance is mostly independent of the particular flavor of architecting process used.

In order to improve the effectiveness of their work in terms of risk and cost control, architects should adhere to the following guidelines:

- *Make risk- and cost assessment part of architectural analysis.* This is a prerequisite to the other guidelines in this list, and implies that architect's skill set should include risk management and cost estimation.

- *Create and maintain a list of architectural concerns and order them by risk and cost.* The top 3-7 items on this list are the most architecturally significant, the concerns that the architect should focus on at any point in time. Apart from helping the architects in their projects, this has the additional benefit of creating a stored history of architectural concerns across multiple projects and architects, which can be analyzed for lessons learned.

- *Regularly monitor the key architectural concerns.* Keep in mind that the architectural significance of a concern may increase after the concern has initially been addressed by architectural decisions.

- *Communicate about architectural concerns and decisions with business stakeholders in terms of risk and cost.* Architects should explicitly link their priorities to the business context of their stakeholders, keeping in mind the purpose of doing architecture in the first place: to manage risk and cost.

- *Get involved in the program's risk register.* This is a special case of the previous guideline, where the stakeholder is the project or programme manager. The architectural concerns all imply risks and hence should be represented in the risk register, and the activities to address the concerns are the associated risk mitigation measures.

- *Report progress in terms of risk and cost control.* The extent to which architectural concerns are under control is a good measure of the progress made during the architectural design phase of a project. The primary deliverables of an architect are the architectural decisions that increase control, and the architect's progress should be tracked on those deliverables (rather than e.g. the chapters of the architectural blueprint).

- *Stop architecting when the impact gets too low.* Spending more resources on addressing concerns than their impact in terms of risk and cost warrants is a waste. Such concerns are clearly not architectural. Don't do more architecture than is strictly necessary [Malan and Bredemeyer, 2002]. Architects invariably have a limited amount of time and they should spend it addressing concerns with the most pressing risks [Fairbanks, 2010] and cost.

8.7 Conclusions and Discussion

In this section, we will discuss related work. We will also briefly discuss a number of questions that were frequently raised when teaching the approach to practicing architects. We will close with our main conclusions.

8.7.1 Related work

Risk in software architecture

Attention to risk is fairly ubiquitous in software development. A state of the art overview of risk management in software development is given in [Bannerman, 2008]. The importance of risk analysis in software development is aptly phrased by Tom Gilb [Gilb, 1988]: "If you don't actively attack the risks, they will actively attack you". Most of this literature does not specifically focus on software architecture. One class of papers and books discusses a variety of risks associated with software development, from requirements volatility to staff turnover. Often, checklists are proposed to systematically investigate a large number of such risks [Boehm, 1991]. Some of the questions posed may relate to the software architecture, such as "Does any of the design depend on unrealistic or optimistic assumptions?" or "Are you reusing or re-engineering software not developed on the project?" [Costa et al., 2007]. Another class of papers discusses sophisticated techniques for computationally handling risks, using Bayesian networks, fuzzy set theory, and the like; [Lee, 1996] is an example hereof. A third type of articles focuses on conceptual models for handling risk in software development. Process models, such as the spiral model [Boehm, 1988], explicitly pay attention to risk analysis as one of the early process steps, to identify areas of uncertainty that are likely to incur risks and next identify strategies to resolve those risks at an early stage. Elsewhere, risk analysis is used to *select* an appropriate process model; for instance, [Boehm and Turner, 2004] uses risk analysis to choose between agile and plan-driven development models.

Attention to risks in software architecture is most prominent in software architecture evaluation. For instance, one of the outputs of the Architecture Tradeoff Analysis Method (ATAM) [Bass et al., 2003] is a list of risks and non-risks. By studying the output of a series of such ATAM evaluations, [Bass et al., 2007] were able to reveal and analyze risk themes specifically geared towards software architecture. [Slyngstad et al., 2008] provide results of a survey amongst software architects to identify risk and risk management issues in software architecture evaluations. One of the lessons they draw is that lack of software architecture evaluation is itself a potential risk.

Risk and cost in decision making

Viewing risk and cost as drivers in architecture decision making has led to approaches like the Cost Benefit Analysis Method (CBAM) [Kazman et al., 2002] and the Architecture Rationalization Method (ARM) [Tang and Han, 2005] that relate architectural decisions to the benefits they bring to an organization, studies that emphasize business implications of architectural decisions [Clements et al., 2007], and approaches that consider architectural decisions as investment decisions [Biffl et al., 2006, Ivanović and America, 2010b].

[Feather et al., 2008] use risk and cost as a driver in requirements decision making. In their defect detection and prevention (DPP) approach, they introduce risk as the primary driver for deciding which requirements to fulfill. Architectural strategies to address the requirements are represented as risk mitigation measures in the model; this may look a bit convoluted, but is fully in line with our view of architecture as a risk management discipline. The cost of (partly) fulfilling a requirement is obtained by adding the cost of all selected mitigation measures for the associated risks.

[Fairbanks, 2010] introduces the Risk-Driven Model, whose aim is to do just enough architecture, based on the risks identified. The method is directed at the overall planning of architectural activities, rather than individual decisions taken during architecting. It also does not treat cost as an explicit factor in prioritizing architectural activities.

Decisions in software development, architecture, buying stock, and many other fields, are made by humans. These decisions often are not purely rational; human decisions are influenced by prior knowledge, time pressure, short term memory, and so on. [Simon, 1969] coined the term *bounded rationality* to denote our limited capabilities for making rational decisions. Sometimes, a third type of rationality is distinguished next to pure rationality and bounded rationality: social/cultural rationalism. There, it is recognized that decision making often is a group process, and the interaction between the decision makers affects the outcome. The different perspectives of the participants may bring new insights and solutions.

When people take decisions, they attach gains and losses to the possible outcomes. If the outcomes are not certain, we may distinguish between four prospects:

1. A high probability of a gain, as in a 95% chance to win $ 1000 (and a 5% chance to win nothing), against 100% chance to win $ 950.

2. A high probability of a loss, as in a 95% chance to lose $ 1000, against 100% chance to lose $ 950.

3. A low probability of a gain, as in a 5% chance to win $ 1000, against a 100% chance to win $ 50.

4. A low probability of a loss, as in a 5% chance to lose $ 1000, against a 100% chance to lose $ 50.

Seminal research by Kahneman and Tversky on this type of decision making has led to what is known as prospect theory, and the fourfold pattern described above [Kahneman and Tversky, 1979, Kahneman, 2011]. It turns out that people behave in a risk averse manner in situations 1) and 4). In situation 1, one prefers a sure gain and does not gamble. In situation 4, one accepts a small loss and does not risk the chance of a large loss. In situations 2) and 3), people behave in a risk seeking way. In situation 2), one tends to gamble and hope for the 5% chance that no loss is incurred. In a similar vein, the hope for a large gain makes people opt for the 5% chance to do so in situation 3). Note that in all four cases, the standard Bernouilli theory results in the same utility for both options ($ 950 in cases 1) and 2), $ 50 in cases 3) and 4)).

The same risk averse/risk seeking behavior is to be expected in decision making in software development projects. One typical example is the continuation of projects that only have a very small chance to ever succeed (situation 2) in the above scheme.

Requirements prioritization

There is a strong resonance between the approach presented here and the extensive literature on requirements prioritization methods (RPMs). The main differences between our approach and RPMs are in the *object* and the *goal* of prioritization:

- RPMs prioritize requirements on the solution, whereas we prioritize stakeholder concerns. These two concepts are related, but they are not the same: concerns are addressed by architectural decisions, requirements are implemented in the solution. One stakeholder concern usually leads to multiple solution requirements, and one requirement can address multiple stakeholder concerns.

- RPMs determine the delivery order of requirements, whereas we help the architect determine the order in which she pays attention to concerns.

A well-known requirements prioritization principle comes from the Agile Manifesto [Agile Alliance, 2001]: “Our highest priority is to satisfy the customer through early and continuous delivery of valuable software”. Thus, many RPMs use business value as the main prioritizing factor [Gilb, 2005], but often other factors are also taken into account, such as return on investment (RoI) [Kazman et al., 2002, Regnell et al., 2008] and risk (mainly in security-oriented requirements engineering [Herrmann et al., 2010]). Our failure scenarios in §8.3.1 are strongly related to the Misuse Cases found

in MOQARE [Herrmann and Paech, 2008]. Based on priority, RPMs allocate requirements to deliver iterations in software projects, or releases in product roadmaps. [Herrmann and Daneva, 2008] examines 15 RPMs, and analyzes their use of benefit and cost as prioritizing factors. [Racheva et al., 2010] derives a conceptual model for requirements prioritization in an agile context based on 19 RPMs. The model identifies five requirements prioritizing aspects for stakeholders: Business Value, Risk, Effort Estimation/ Size Measurement, Learning Experience, and External Change. We recognize the two main themes of this chapter: Risk and Cost. For a discussion on Business Value in architecting, please refer to the next section. For architects, the other two factors are an integral part of their architecting process: External Change is usually handled as a modifiability concern and a risk factor, while Learning Experience is the means by which architects address uncertainty in the solution domain, using practices like architectural prototyping.

Architects should be aware of both the differences and the similarities between RPMs and risk- and cost based architecting. The similarities mean that the RPM prioritization techniques can help architects prioritize their concerns. The differences are equally important: an architect should not focus exclusively on high priority requirements in terms of delivery order, since requirements scheduled for later delivery may have high impact on the architecture, and prove very risky if ignored. As stated in [Abrahamsson et al., 2010], "certain classes of systems that ignore architectural issues for too long hit a wall and collapse due to a lack of an architectural focus."

8.7.2 Frequently Asked Questions

The topics in this section were raised by the architects we trained in the Risk- and Cost Driven Architecture approach. Since they led to interesting discussions, we are presenting them here.

What about existing systems?

What does our view on architecture mean for existing systems, i.e. systems after their initial delivery? Since we already know how the architectural decisions made during the design phase turned out, does it still make sense to talk about the risk of making wrong decisions? It does: just like during the design phase, the architectural activities related to existing systems have risk and cost management as their prime business objective. Typical activities at this stage are architecture recovery, evaluation and architectural modifications to the system. The reason an architect gets involved is to identify architecturally significant concerns related to something that the stakeholders want to do with this system; most likely, modify it in some way. And once again, what is archi-

tecturally significant is determined by risk and cost impact, and decisions need to be made to address these concerns, e.g: do we refactor a component that is hard to change? Do we port the system to another platform? [Klusener et al., 2005], in their extensive paper on modifying existing systems, come to a very similar notion of architectural significance in those systems, as we have seen in §8.3.1. [Slyngstad et al., 2008] have surveyed risks in software architecture evolution, and present the most common risks for architectures of existing systems and their associated mitigation activities.

What about value?

One might argue that a solution's value to its stakeholders should play at least as important a role as the risk and cost of delivering it. In practice, we find that solution architects are less concerned with stakeholder value, especially when they operate in a project context. This is because most value considerations have already been taken into account in the solution's goals and business requirements, which serve as input to the project. This process of pre-determining the value of a solution by fixing its high-level requirements is usually considered part of the requirements analysis rather than the architecting phase of a solution's lifecycle, and is often largely completed even before the solution architect gets involved. A frequently occurring example of this situation is when a supplier architects a solution in response to a Request for Proposal (RfP): the RfP documentation contains requirements that encapsulate the requested solution's business value. As long as the architected solution fulfills these business requirements, the value objective is considered to be fulfilled, and it is the architect's job to fulfill the RfP requirements at the lowest possible cost. We even see that solutions that add stakeholder value beyond the previously captured requirements are often regarded with suspicion. The management jargon for this situation is "gold-plating", and it has strong negative connotations. However, as we have seen in Chapter 3, there are dangers associated with determining value-based requirements without taking the architecture into account.

In practice, there are two types of situations where solution architects are involved in stakeholder value discussions:

1. When the solution architect is involved in the analysis work. A good example of this is the Cost Benefit Analysis Method [Kazman et al., 2002], a method for performing economic modeling of software systems, centered on an analysis of their architecture. Another example is the requirements convergence planning practice presented in Chapter 3.

2. Creating value for "internal" stakeholders in the delivery project, such as the developers and the project manager. Examples are architectural decisions that

create re-usable components or make the solution's construction more efficient. In this situation, the value actually consists of cost savings, reinforcing the point that the architect's work is cost-driven.

In short, when architectural features or requirements are prioritized in order to determine *what* to build in a solution, value plays an important role. The focus of solution architecting in this chapter, however, is on *how* to build a solution, and then risk and cost trump value.

As explained above, [Kahneman, 2011] finds that people are prepared to take high risks if there is a chance for a high gain, even if such a choice is not rational. Our approach raises the profile of risks and costs in the trade-off against value, and one would expect that this would help to prevent irrational architectural decisions in high-risk situations. It would be interesting to validate this expectation by presenting architects with high-risk, high-gain architectural decision scenarios and analyzing differences in responses depending on their training and their knowledge of expected short-term gains.

Does this mean architects always have to minimize risks?

Risk and cost are used to assess the architectural significance of concerns, and should play a role in trade-offs between decisions addressing these concerns. This does not automatically mean that architects or stakeholders should always select the architectural alternative with the lowest risk: that is up to them entirely, and depends on other factors such as the risk-averseness of the culture in their organization. What it *does* mean is that these risks should be made explicit and considered in the trade-off.

8.7.3 Conclusion

We have presented and elaborated a view of solution architecting as a risk- and cost management discipline. Although this view is an extension of pre-existing views on software architecture, it goes beyond software architecture alone: it includes other architecture genres, captured under the name Solution Architecture as described in §1.2.1.

The risk- and cost driven view on architecture is the basis for the architecting approach presented in the following chapter: Risk- and Cost Driven Architecture. It is part of a solution architecture training programme that has so far been taught to 159 architects, many of whom have claimed that it helps them become more effective in their jobs. These claims are supported by anecdotal evidence, some of which we have presented here. In Chapter 9, we will further substantiate the claims.

In conclusion, viewing architecture as a risk- and cost management discipline helps architects and stakeholders in focusing their activities on high-impact concerns, and in doing so raises the value of architecture to organizations.

9

Risk- and Cost Driven Architecture: a Pragmatic Solution Architecting Approach

This chapter describes RCDA, the solution architecting approach we developed in Logica. The approach consists of a set of practices, harvested from 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. The survey shows that for the majority of trainees, RCDA has significant positive impact on their solution architecting work.

9.1 Introduction

In Chapter 6, we described how in 2006, we started out to create a generic architecting process for Logica. The result of this effort is presented in this chapter. It is an integrated set of solution architecting practices, collectively called Risk- and Cost Driven Architecture (RCDA). We first give a short summary of the principles and practices that make up RCDA. We then explain the structure that integrates the practices, including the rationale behind it, and explain how the approach was implemented among the company's architects. We then present the result of a survey measuring the effect of RCDA training on the architects.

9.2 The RCDA Approach

In this section, we first describe the practices that are the basic building blocks of RCDA. We then explain the four key principles the approach is based on, and the way the approach is implemented in the organization. We conclude with a clarification and justification of the structure of RCDA: why it uses practices, what elements exists beyond the practices, and what structures are used to organize them.

9.2.1 RCDA practices

The basic building blocks of RCDA are *practices*. A practice is a way to systematically characterize a problem and address it. The practice concept will be explained further in §9.2.4. Practices that address closely related problems are clustered into practice sets. There are practice sets for Requirements Analysis, Solution Shaping, Architecture Validation, Architecture Fulfillment, Architectural Planning and Architectural Asset Management.

The practices of RCDA, ordered by practice set, are:

Requirements Analysis practices, where the requirements originating from stakeholders are prepared for shaping a Solution:

- **Architectural Requirements Prioritization** addresses the problem of pinpointing architecturally significant requirements and concerns, according to the principles laid out in Chapter 8.
- **Dealing with Non-Functional Requirements** gives guidance on handling NFRs, which are often underexposed and can have major impact on the solution; it contains the key elements from Part I of this thesis.
- **Stakeholder Workshop** is a practice for obtaining architectural requirements from stakeholders, based on the SEI's Quality Attribute Workshop [Barbacci et al., 2003].

Solution Shaping practices to define a solution's architecture:

- **Solution Selection** addresses the problem how to identify and select the best fitting strategy to fulfill architectural requirements on a Solution in an objective manner, implementing one of the lessons learned from Chapter 5.
- **Solution Shaping Workshop** is a special case of a Stakeholder Workshop. All Logica stakeholders are gathered to kick start the solution shaping process, led by the solution architect.

Cost-Benefit Analysis helps architects to consider the return on investment of any architectural decision and provides guidance on the economic tradeoffs involved, based on the SEI's CBAM practice [Kazman et al., 2002].

Applying Architectural Strategies describes how to implement architectural strategies selected in previous steps, determining a solution's structure according to the principles explained in e.g. [Bass et al., 2003, Gamma et al., 1995].

Architecture Documentation documents the current state of the solutions architecture in a set of views [Kruchten, 1995, ISO 42010, 2011], focussed on effectively communicating the architecture to the relevant stakeholders.

Documenting Architectural Decisions addresses the problem of tracking architectural concerns and decisions throughout their lifecycle, based on e.g. [Tyree and Akerman, 2005, Jansen and Bosch, 2005].

Solution Costing gives guidance on early costing of delivering a solution using a selected architecture.

Architecture Validation practices aimed at validating the architecture developed in previous steps:

Architecture Evaluation to create transparency and identify risks in the architectural decisions made, and to verify that the architecture meets its requirements; roughly based on [Abowd et al., 1997]

Architectural Prototyping is performed when there is uncertainty about the feasibility of (parts of) an architecture which can be resolved by "trying it out".

Supplier Evaluation helps architects identify potential risks when committing to delivering third party products as components of an architected solution.

Architecture Fulfillment practices, related to the development and delivery of the solution under architecture:

Architecture Implementation making sure that the architecture developed and validated in previous steps is actually implemented in the solution.

Architecture Maintenance provides guidance on taking an existing solution into operation, and on maintaining a solution's architecture once it is operational.

Blended Architecting gives guidance on solution shaping and fulfillment in a geographically distributed solution delivery setting.

Architectural Planning practices, giving guidance on how to plan architecting activities:

Architecting Lifecycles addresses the problem of when to apply RCDA practices, showing how the various RCDA practices map to certain common scenarios.

Requirements Convergence Planning addresses the problem of finding the best balance of affordability between cost and benefit of architectural requirements, as explained in Chapter 3.

Architecture Contingency Planning helps mitigate the risk of having to backtrack architectural decisions when it turns out an architecture cannot fulfill the stakeholders' needs.

Architectural Asset Management practices, aimed at re-using architectural assets like knowledge, reference architectures and re-usable components across solutions:

Architecture Knowledge Management addresses the problem of codifying and sharing architectural knowledge such as patterns and lessons learned across the company, using techniques like those documented in [Zimmermann et al., 2007] and [Farenhorst and van Vliet, 2008].

Software Product Line Management gives guidance on how to implement and manage software product lines so that they serve as the basis for multiple solutions, based on the SEI Software Product Line materials [Clements and Northrop, 2002].

Technology Monitoring helps architects keep abreast of new developments that can provide more fitting alternatives for solution selecting.

The practices, grouped in practice sets, are visualized in Fig. 9.1.

9.2.2 RCDA principles

Risk- and Cost Driven Architecture is based on the following key principles:

- Cost and Risks drive architecture.
- Architecture should be minimal.
- Architecture as both Blueprint and Design Decisions.
- Solution Architect as Design Authority.

These principles are based on our experiences in technical assurance (see §1.1), enhanced by literature. Solution architects are encouraged to always keep these principles in mind when applying RCDA practices. The principles are applied through the individual RCDA practices as explained below.

The first principle, *Cost and risks drive architecture*, is explained extensively in Chapter 8. It is applied throughout RCDA, but most explicit in the Architectural Requirements Prioritization practice.

Architecture should be minimal is based on recent insights such as expressed in [Malan and Bredemeyer, 2002] and [Fairbanks, 2010]. In order to keep overview of the whole system, the solution architect's decisions should be limited to those that have critical impact on the system and its delivery - leaving a maximum of design space for filling in details within the constraints set by that architecture. This should of course be done with due consideration for the capabilities of those designers and developers, and should not detract from the clarity with which the architecture is communicated. Kazman, Bass and Klein formulate this principle as: "A software architecture should be defined in terms of elements that are coarse enough for human intellectual control and specific enough for meaningful reasoning." [Kazman et al., 2006] It is applied through the Architectural Requirements Prioritization practice.

Architecture as both blueprint and design decisions is based on the second view on architecture described in §1.2.1 and papers such as [Jansen and Bosch, 2005, Tyree and Akerman, 2005]. The architecture of a system is more than just a blueprint of its high-level structure - the design decisions leading to that structure and the underlying rationale are equally essential. No architectural description is complete without a well-documented set of design decisions. By thinking about architecture as a set of design decisions, we abstract away from the modeling details inherent to a particular technology or view, and are able to give generic guidance on how architects make trade-offs and document decisions. It also helps to focus on the rationale behind the decisions, which is important to future architects and those implementing or reviewing the architecture. This principle is applied through the practices Architectural Requirements Prioritization, Solution Selection and Documenting Architectural Decisions.

Solution architect as design authority is based on views like those documented in [Fowler, 2003] and [Clements et al., 2007]. The complexity of today's IT solutions requires that the most critical design decisions are made by one person with an overview of the whole system. This person should have the authority and the subject matter skills and knowledge to make such decisions. This role is distinct from the project manager's role, and is called the Solution Architect in RCDA. Of course, architecture is often team work, and architects should surround themselves with experts to help them make critical decisions - but in the end, no matter how big the design team, one person is responsible for making all the trade-offs and the final decision. This principle is applied

through the RCDA Solution Architect role.

9.2.3 Implementation

RCDA is documented on a company intranet site. On the RCDA site, each practice is described according to a defined structure, with the following sections: Objectives, Approach, Roles, Input, Activities, Output. Next to the practice descriptions, there are web pages explaining the principles, key concepts, roles and templates. These is also a quick reference guide and an "about" section that explains the background and future plans.

The current version of RCDA is 1.1. It was reviewed and ratified by an international panel of representatives from every major company cluster and country. It is embedded into the company's business operating model as the recommended solution architecture approach.

The RCDA approach underpins the company's internal training program for solution architects. The core of this program is the Solution Architecture Practitioner Course. In 2010 and 2011, a total of 159 architects were trained in RCDA.

9.2.4 Structure of RCDA

As described in Chapter 6, we started out to create a generic architecting process in 2006. We identified a number of business goals and usage scenarios to scope the process (§6.2.2), and documented requirements that the process we were creating had to fulfill, based on the business goals (Table 6.1)and the CMMI maturity level 3 objective (Table 6.3). Once the requirements were clear, it took us about a month to write a 60 page draft process description – and then we got stuck. It turned out that the scalability and flexibility requirements (*rq.scalable* and *rq.generic*) were too much to be accommodated by a single process description. [Kazman et al., 2006] mentions "component techniques" for architecture analysis and design that can be "combined in countless ways to create needs-specific methods in an agile way"; we had harvested a number of such techniques, but a traditional process description did not allow us the agility required. We put the issue aside until we could find a way to resolve it.

We found a solution in 2008, when we came across Ivar Jacobson's practices approach [Jacobson et al., 2007]. Jacobson identifies a number of problems with traditional process descriptions, that touch the core of our issues in designing a generic architecture process:

Problem of Completeness "By striving for completeness, the processes end up as brittle, all-or-nothing propositions." We had tried to construct a complete archi-

tecting process for all possible business scenarios, and ended up with a document that made it hard for our architects to identify the parts and techniques that would add value to their specific situations. We also had many pieces of guidance that we wanted to share with our architects, but which were waiting for the process description to be complete before they could be released.

Problem of Adopting a Complete Process "Each [...] team has its own way - of - working (explicit or tacit), changing everything is silly, changing one thing may be smart." We wanted our architects to improve their existing processes to improve their architecting practices, rather than completely replace them with a heavy new architecting process.

Problem of Acquiring Knowledge "People don't read process manuals or language specifications, they want to apply processes not read about them." We realized our architects didn't need a 60 page detailed process description: they needed easily digestible, bite-sized pieces of guidance that would help them deal with their specific problems in their specific contexts.

Jacobson introduces an alternative to the process description: the *practice*, a "way to systematically and verifiably address a particular aspect of a problem." We decided to adopt this alternative for our solution architecting approach. The key aspects of the practice approach we adopted for RCDA are:

- Practices describe a way to characterize a problem and a way to address it.
- Practices can be picked and applied individually or in combination with each other to fit a particular situation.

The practices approach helped us address a number of challenges that we had run into when writing a traditional process description:

- It allowed us to disseminate important guidance without first having to document a complete, extensive process, solving the problem of completeness.
- It allowed architects to easily find relevant guidance without being forced to read a whole process description, partly solving the problem of acquiring knowledge.
- It allowed architects to apply individual practices in digestible bites, and change only those aspects of their way of working that would add value, solving the problem of adopting a complete process.

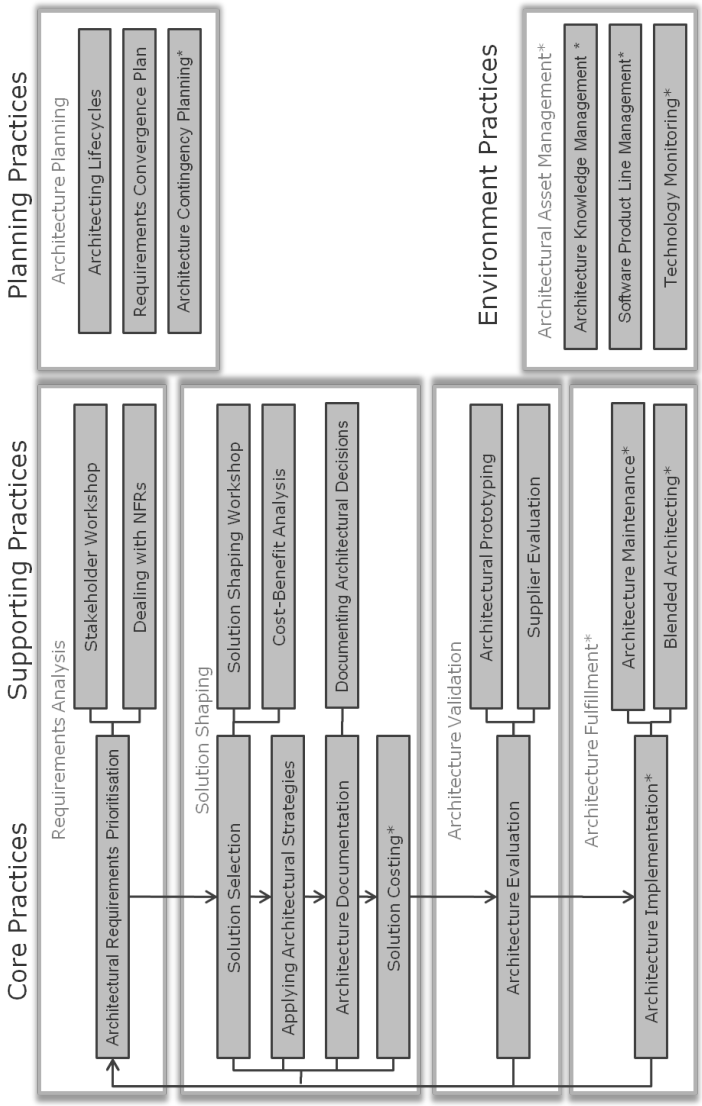


Figure 9.1: RCDA Practices organized by practice set and practice category.

Two years after adopting the idea of working with practices, we rolled out the result: the first version of our solution architecture approach, called Risk- and Cost Driven Architecture (RCDA). In 2011, we rolled out RCDA 1.1, the current version. RCDA 1.1 consists of 14 practices.

Architects are not normally expected to use all of the practices all of the time: they need to identify the best fit practices for their specific context. In order to help architects select and navigate through the practices of interest to them, we created a number of dimensions to structure the collection of practices:

Practice sets group practices into related tasks with similar objectives, as described in §9.2.1

Architecting lifecycles are typical real-life architecting scenarios, indicating which practices are used when; they are documented in the Architecting Lifecycles practice

Practice categories divide practices into four categories:

Core practices The downside of moving from a traditional process description to a practice approach was that a set of practices by itself cannot fulfill the CMMI compliance requirement for a defined process (*rq.cmmi.gen* in Table 6.3). We resolved this by chaining the seven RCDA core practices together to form a core architecting process that should be followed in every reasonably complex project. The core process is one of the scenarios documented in the Architecting Lifecycles practice. Core practices may refer to supporting practices for (optional) additional guidance. The core practices embody the principles of RCDA.

Supporting practices Supporting practices provide additional guidance on good architecture in a project, product or bid context. This category contains all non-core practices in the solution domain, except planning practices.

Planning practices Planning practices help the architect and project / bid manager to plan architecture activities. They are all the practices in the Architecture Planning practice set.

Environment practices Environment practices are architecture practices in a bid, project or product's environment that provide and consume artifacts of the solution domain, and in general impact the solution architecting, but are not solely directed at one solution. Thus, environment practices are about architecting across multiple individual solutions. At the moment the only environment practice set in RCDA is Architectural Asset Management; in

the future, additional environment practice sets may be added, related to e.g. architectural quality monitoring.

These structures are visualized in Fig. 9.1. The figure shows 22 practices: 14 that are part of RCDA 1.1, and 8 that have been identified for future versions. The 8 future practices are marked with an asterisk. In the figure, the 7 core practices are chained together by arrows to symbolize the core architecting process that they form.

9.3 Impact Survey

In October 2011, all Logica architects that were trained in RCDA were surveyed. The objective of the survey was to assess the impact of RCDA and its training on the work of the architects. In addition to the survey itself, we organized an expert workshop; a guided discussion with a select group of RCDA trained architecture experts. The workshop was held after the survey, and its purpose was to enhance the initial quantitative analysis results with qualitative knowledge from practicing architects.

9.3.1 Survey description

At the time of the survey, 159 people were registered as having received RCDA training. All of these registered trainees received an invitation by e-mail to participate in the survey. After two weeks, 32 (20%) had completed the survey, and the survey was closed.

The survey consisted of three sections:

Section A General questions about the trainees' activities after the training.

Section B Specific questions asking the respondents about the impact and frequency of use of the guidance in RCDA.

Section C Questions asking respondents whether they agreed with statements on the overall effectiveness of RCDA.

In order to measure at the level of individual pieces of guidance in RCDA, we codified the most important guidance: Table 9.1 lists the practices in RCDA 1.1. We have distilled one or more key guidance elements from each practice. Every guidance element has been given a code tag, which is used to identify the guidance element in the survey.

In Section B, respondents were asked to indicate on a Likert scale how often they had applied each guidance element in Table 9.1 both before and after receiving the training:

Table 9.1: RCDA practices and key guidance elements.

Architectural Requirements Prioritization	
ARP.rc	Identify architectural requirements by risk and cost impact.
ARP.sc	Express architectural requirements in scenarios.
ARP.wf	The architect's daily workflow is addressing architectural concerns, prioritized by risk and cost impact.
Dealing with Non-Functional Requirements	
NFR.hd	Look for hidden NFRs, since they are often crucial for acceptability even when not documented.
NFR.vf	Verify as early as possible that the architectural design will fulfill the NFRs.
NFR.cm	Don't commit to quantified NFRs until you have proof of feasibility.
NFR.dc	Document how NFRs are dealt with as proof of professional behavior.
Stakeholder Workshop	
SW.ws	Gather stakeholders in a workshop to elicit architectural requirements as early as possible.
Solution Selection	
SS.ev	Decide after evaluating multiple alternative solutions against objective criteria.
Solution Shaping Workshop	
SSW.ws	At the start of a bid or project, gather all delivery stakeholders in a solution shaping workshop to agree on a candidate solution.
Cost-Benefit Analysis	
CBA.qf	Quantify the impact of architectural strategies on a solution's quality attributes in terms of stakeholder value.
Applying Architectural Strategies	
AAS.dc	Document the impact of selected architectural strategies in terms of elements, interfaces and refined requirements.
AAS.rp	After applying strategies, re-prioritize architectural concerns.
Architecture Documentation	
AD.sa	Use a stakeholder analysis to determine to whom the documentation is communicating.
AD.vp	Use viewpoints to show stakeholders how their concerns are addressed.
Documenting Architectural Decisions	
DAD.rd	Use a formal Record of Decision to document key architectural decisions.
DAD.rg	Use an architectural concern and decision register to prioritize and order the architecture work.
DAD.pr	Communicate progress and status of architecture work in terms of architectural concerns and decisions.
Architecture Evaluation	
AE.ev	At key points in an architecture's lifecycle, perform an objective evaluation and analysis of how the architecture fulfills its stakeholders' needs.
Architectural Prototyping	
AP.pr	When necessary, build a prototype or proof-of-concept to verify that architectural strategies fulfill the requirements.
AP.oc	Prepare to deal with any outcome of the PoC, including a contingency plan in case of a negative result.
Supplier Evaluation	
SE.ev	When third parties provide critical components of our architectural solution, evaluate the supplier to identify potential commercial, technical, PR, quality and service related risks.
Requirements Convergence Planning	
RCP.pl	In case of unfeasible or unclear NFRs, agree a plan with the client that describes how to converge on acceptance criteria, representing a balance of affordability between cost and benefits.
Architecture Lifecycles	
AL.cp	RCDA Core Process.
AL.rf	Respond to RfP.
AL.ru	RUP software development.

- never
- once or twice
- regularly (whenever an applicable situation occurs)
- every day (part of my daily work routine)

Respondents were also asked to indicate the impact of the guidance by choosing between:

- n/a (never applied the guidance)
- counter-productive (I tried applying the guidance, but it made matters worse)
- neutral / mixed results
- noticeable improvement (compared to acting without this guidance)
- critical improvement (without applying this guidance, project would have failed or bid would have been lost)

9.3.2 Survey results

Some general statistics to start with:

- Elapsed time from the training to the survey was between 3 and 23 months, with an average of 9 months, meaning all respondents had time to internalize and apply the material.
- Average time spent in architect roles was 45%. 6 respondents spent less than 10% in architect roles, of which 5 indicated they had not been in any architect role. 12 respondents spent 75% or more of their time in architecting roles.
- 13 respondents (40%) were the lead architect on the majority of their assignments, meaning they were responsible for architectural decisions.

Fig. 9.2 shows the responses to the three general statements about the effectiveness of RCDA (Section C):

imp_gen *In general, my effectiveness as an architect has improved after being trained in RCDA.*
imp_com_stkh *RCDA helps me to communicate with stakeholders more effectively.*
imp_prio *RCDA helps me to better focus and prioritize my work as an architect.*

Overall, the majority of the responding architects agree that their effectiveness has increased, and about half agree that RCDA has brought them the benefits of the risk- and cost driven approach described in §8.1. Less then 15% disagree with any of the statements.

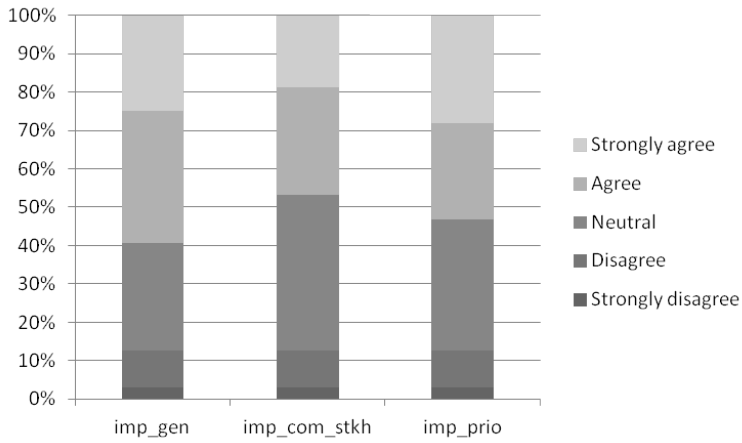


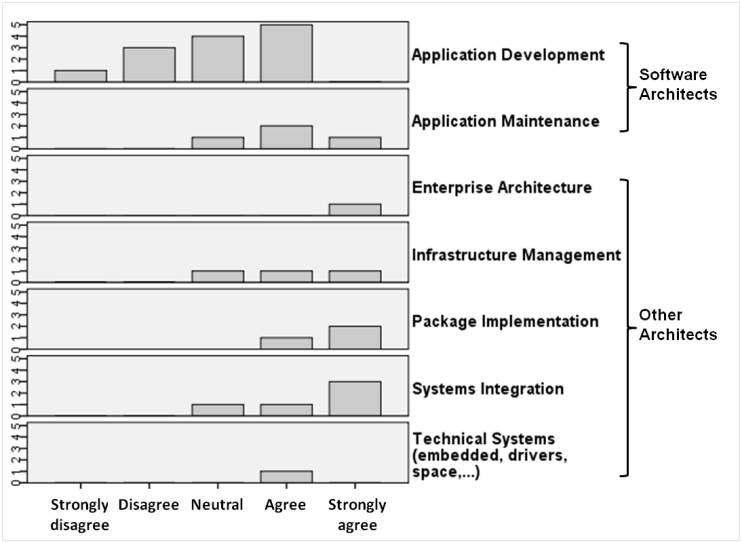
Figure 9.2: Three statements on effectiveness of RCDA

In Fig. 9.3(a), we see how the responses to imp_gen are divided over the various architecture "genres". The figure confirms that the effectiveness of the risk- and cost driven view extends beyond software architecture into the wider domain of solution architecture. In fact, the only disagreement comes from the software architects in the application development domain. The fact that only some *software* architects disagree with the effectiveness of RCDA seems remarkable, since RCDA is mostly based on ideas from the software architecture community. We discussed this paradox in the expert workshop; the most likely explanation seems to be that some software architects may have found RCDA less value-adding than other architects, because it is partly based on ideas that were already familiar to them before receiving the training.

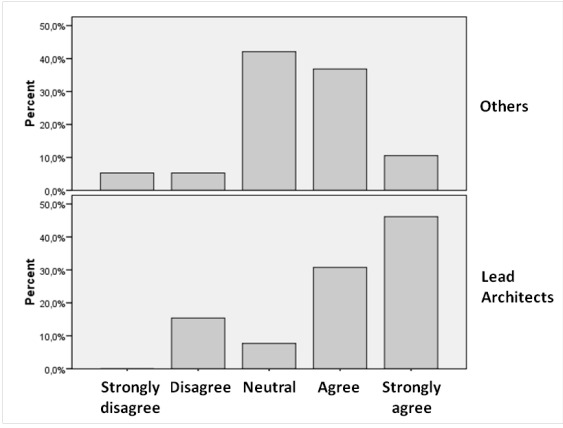
Fig. 9.3(b) shows that those who have been active in lead architect roles have stronger opinions, and in general are more positive about RCDA effectiveness than those who have not been in lead architect roles. This visual impression is confirmed by statistical analysis, which shows that the "lead architect" responses are significantly correlated to the "general effectiveness" agreement responses (Spearman's ρ correlation coefficient of 0.34, 1-tailed significance at the 0.05 level).

RCDA principles

We asked the architects about the frequency with which they applied the general RCDA principles explained in §9.2.2, and the impact. Table 9.2 shows the results. Column



(a) By architecture genre



(b) Lead architects vs others

Figure 9.3: Agreement with "In general, my effectiveness as an architect has improved after being trained in RCDA."

Table 9.2: RCDA Principles: frequency applied and impact

Principle	Applied before	Applied after	Significant impact
Cost and risks drive architecture*	16%±6%	34% ±8%	86%±7%
Architecture should be minimal*	34%±8%	53%±8%	88%±6%
Architecture as both blueprint and design decisions	34%±8%	53%±8%	100%
Solution architect as design authority*	25%±7%	44%±8%	86%±7%

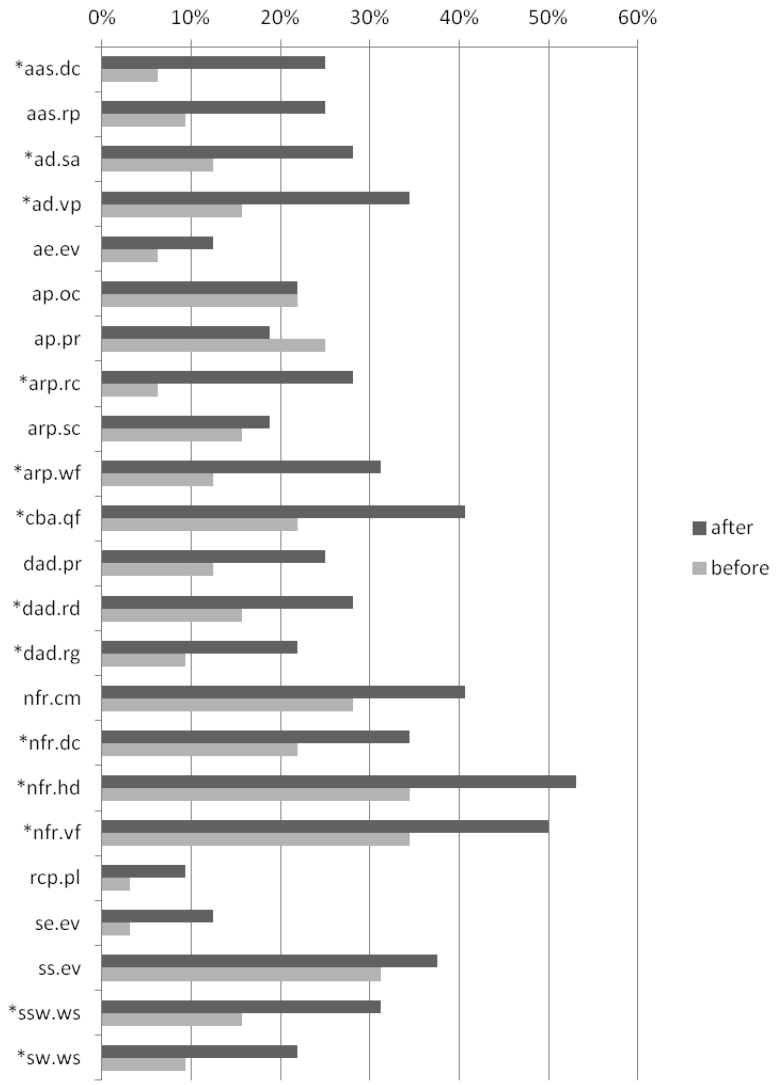
* Increase in application frequency significant at 0.05 level

"Applied before" and "Applied after" shows the percentage of respondents who applied the principle before and after receiving RCDA training; "Impact" shows the percentage of respondents who reported significant impact. Standard errors in the percentages are indicated in the table. The table shows that the number of architects applying the principles has increased considerably after the training, and that all four of the principles have significant impact when applied. A paired-sample T-test between the "frequency applied before training" and "frequency applied after training" shows that three out of the four principles have been applied significantly more after receiving the training: the increase in application of the "architecture as both blueprint and design decisions" principle is not significant at the 0.05 level, the other three are significant and are indicated with an asterisk.

RCDA practices

Fig. 9.4 shows the percentage of respondents indicating they have applied the key guidance elements of the RCDA practices listed in Table 9.1, both before and after receiving the training. All guidance elements show an increased number of respondents applying it after the training, with the exception of Architectural Prototyping. The guidance elements for which the increase is significant as calculated by a paired-sample T-test are indicated with an asterisk. The Architectural Prototyping practice was already applied before the training by 25% of respondents.

The percentage of trainees who applied the guidance after training is below 60% for all guidance. This may seem low; additional light is shed on this if we compare the percentages for lead architects versus other respondents, as visualized in Fig. 9.5. We see that more lead architects than others are applying the guidance, and almost half of the guidance elements are applied by the majority of the lead architects. Analysis shows that the "lead architect" responses are significantly correlated to the "frequency applied after training" responses for all guidance elements except ap.pr and nfr.cm



* Increase in application frequency significant at 0.05 level

Figure 9.4: RCDA practices: respondents applying guidance before and after training (abbreviations on p.157)

(positive correlation coefficient of 0.3 or more, 1-tailed significance at the 0.05 level, using Spearman's ρ). Just like in the case of "general effectiveness" above, we see a correlation between the solution architect's position of authority and responsibility (lead architect) and the frequency of applying RCDA guidance.

Fig. 9.6 shows the key guidance elements of the RCDA practices. The figure visualizes the impact of the practices and their training. The horizontal axis represents the percentage of respondents reporting an increased frequency of applying the guidance after the training. The vertical axis represents the percentage of respondents reporting that the guidance has had significant positive impact in their projects. The first observation is that none of the practices is reported to have increased application by more than 50% of respondents (which is in line with Fig. 9.4). On the other hand, all practices are reported to have significant impact by over 50%.

We have clustered the guidance elements and separated the clusters by gray lines.

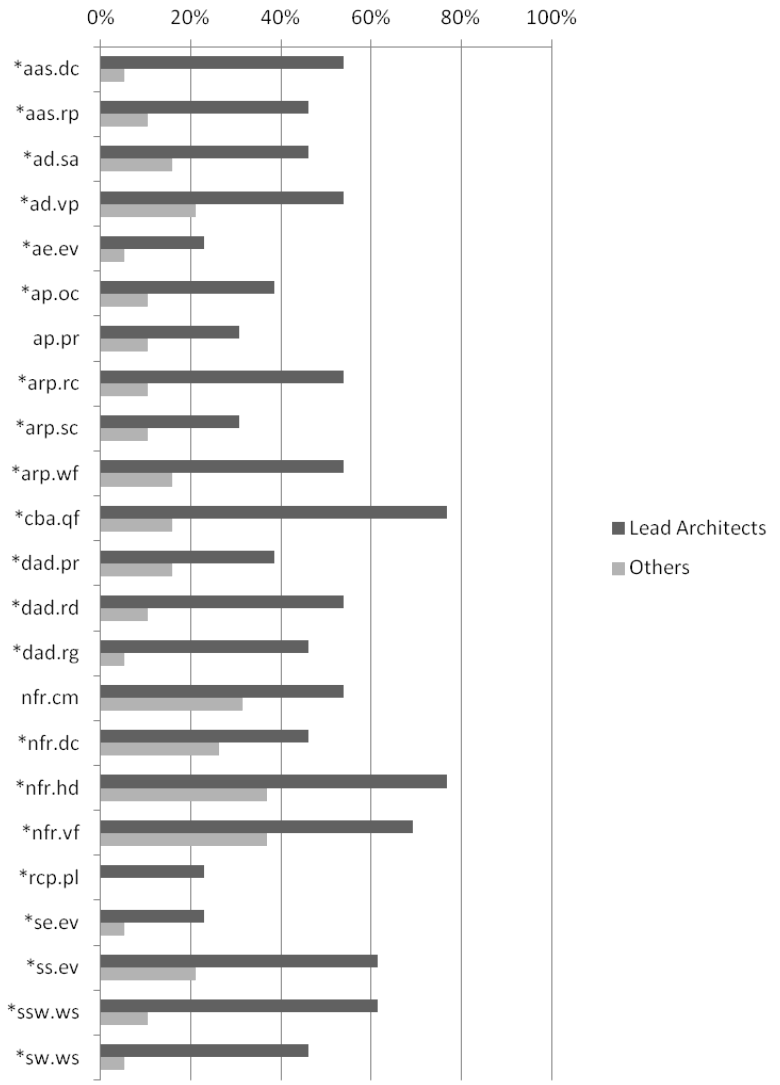
- In the center, we see 8 guidance elements that all have around average characteristics: increased application by about 25%, and significant impact reported by about 75% of respondents. This cluster includes Architecture Evaluation, Cost-Benefit Analysis, Applying Architectural Strategies, Stakeholder Workshop and guidance elements from three other practices.

- In the top left cluster, we see both the Architectural Prototyping and the Requirements Convergence Plan practices. Apparently, the use of these practices has not increased very much, even though their impact is relatively high. Part of the explanation for this could be that both of these practices require considerable resources and time to implement.

- In the top right cluster are the "stars" of the training: the guidance elements that have the highest impact in terms of both usage and effectiveness. This cluster contains the Solution Shaping Workshop, most of the guidance from Dealing with NFRs and Documenting Architectural Decisions, and the use of viewpoints in Architectural Documentation.

- The bottom left cluster has Supplier Evaluation and Solution Selection, two practices that require relatively formal evaluations to be performed. These are perceived as relatively low-impact practices by the architects.

- The bottom right cluster contains all guidance in the Architectural Requirements Prioritization practice. The training appears to be relatively successful in making architects consciously prioritize their requirements; on the other hand, "only" around 70% of the architects report that this has significant impact. A possible explanation came out of the post survey expert workshop: because requirements



* Correlation with lead architect response significant at 0.05 level

Figure 9.5: RCDA practices: lead architects vs others applying guidance after training (abbreviations on p.157)

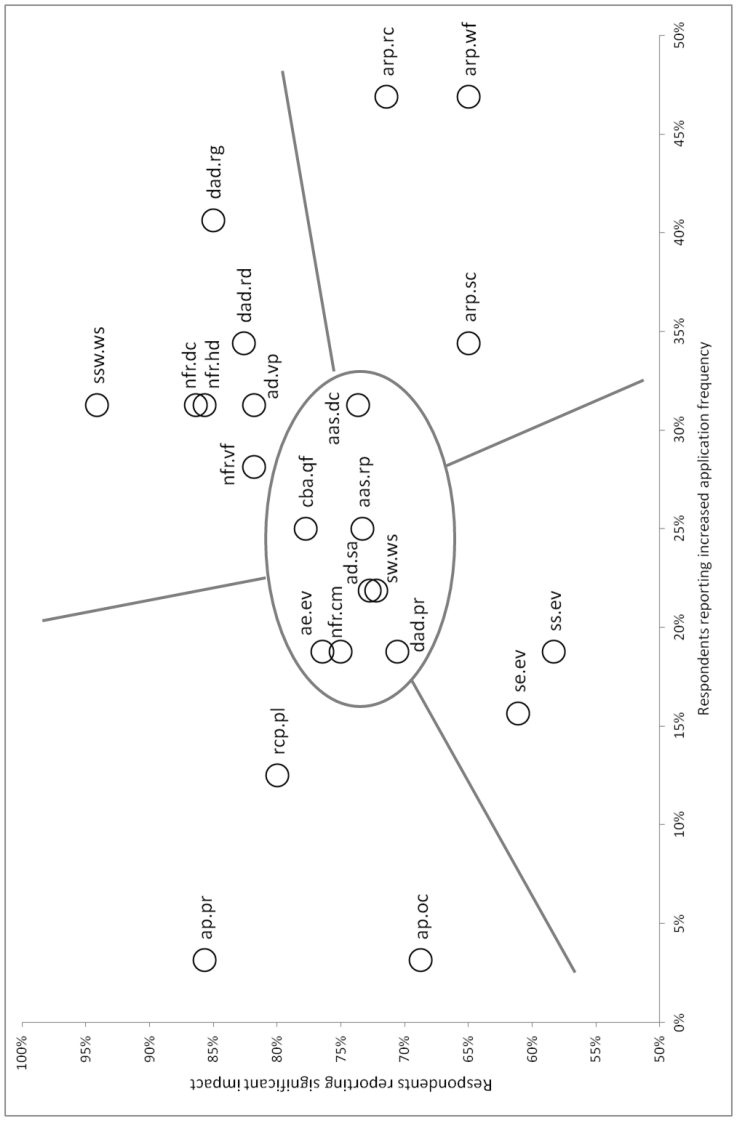


Figure 9.6: RCDA practices: impact vs. increased application after training (abbreviations on p.157)

prioritization happens relatively early in the chain of architecting activities, its impact is perceived as more indirect than that of other practices.

Due to the limited size of the population sample, the standard error in the placement of the guidance elements is in the same order of magnitude as the separation between the clusters: it ranges between 6% and 8% on the x-axis and between 5% and 11% on the y-axis. This means that the clustering should be considered tentative at this point.

Because we found significant differences between the responses of lead architects versus other architects for the frequency and effectiveness questions, we also looked for such differences in the impact responses for the individual guidance elements (the Y-axis in Fig. 9.6). We found only one: all 10 lead architects (100%) who applied architectural prototyping (ap.pr) reported significant impact, while of the 11 other architects who applied ap.pr, only 8 (73%) reported significant impact. Analysis shows that the "lead architect" response is significantly correlated to the "impact" response for ap.pr (positive correlation coefficient of 0.435, 2-tailed significance at the 0.05 level, using Spearman's ρ).

9.4 Conclusions and Discussion

The results of the analysis above indicate that for the majority of trainees, RCDA has significant positive impact on their solution architecting work. This is true for RCDA as a whole, for its principles, and for its individual practices. The RCDA training is effective in increasing the application of the principles and practices.

The survey validates RCDA as an approach that improves the solution architecting practice in Logica.

The sparse application and relatively low appreciation of formal evaluation practices like se.ev and ss.ev (Fig. 9.6) is in line with findings by e.g. [Clerc et al., 2007], which reports that "methods and techniques to validate the architecture . . . are not embedded within the mindset of architects." Another finding from the [Clerc et al., 2007] survey is that "the architects mindset lacks focus on reflections on those decisions as building blocks for software architectures"; the success of the RCDA Documenting Architectural Decision practice in terms of both frequency of use and perceived impact indicates that this lack of focus can be remedied by e.g. the RCDA training.

In the after-survey expert workshop, we discussed some of the more remarkable results of the survey with a selected group of senior architects who were familiar with RCDA and its training. The results of this workshop are discussed below.

Architectural prototyping

The prototyping guidance element "When necessary, build a prototype or proof-of-concept to verify that architectural strategies fulfill the requirements" (ap.pr) jumps out in a number of results:

O1 ap.pr is the only guidance element less frequently applied *after* training than *before* training.
O2 ap.pr is one of only two guidance elements whose application frequency after training is *not* significantly correlated with the lead architect role.
O3 ap.pr is the only guidance element whose impact response *is* significantly correlated with the lead architect role.

The workshop participants produced two possible explanations for the decrease in application after training (O1):

E1 RCDA focuses architects' attention on other activities, making prototyping a relatively lower priority.
E2 The time passed after the training is less than before the training, the architects simply didn't have enough time after the training to apply ap.pr, which require considerable resources and time to implement.

Taking all three observations together, the workshop agreed that E2 is the more likely explanation, since E2 helps explain O2, and E1 does not match with O3. E2 also is a good explanation for the fact that in Fig. 9.6, ap.pr is in the top left cluster with requirements convergence planning, another practice that requires significant planning and use of resources.

Lead architect

The trainees that were in lead architect roles after the training have given significantly more positive responses to most of the questions related to application frequency and overall effectiveness. The post survey expert workshop generated a number of explanations for this phenomenon:

L1 Those in the lead determine which practices will be followed, so they can choose to apply RCDA practices, while those not in the lead have to follow practices dictated by others.
L2 The use of a common approach like RCDA is much more important for those in the lead, since it smooths communication with stakeholders like reviewers and managers – with whom those not in leading roles have less dealings.

L3 RCDA promotes a position of authority for architects (the fourth principle in §9.2.2), so that those who apply RCDA tend to take more ownership and responsibility, which puts them in leadership positions.

The data set did not provide any means to confirm or reject any of the three explanations: they may well all be valid, and reinforce each other's impact.

9.4.1 Threats to validity

In a survey like this, there is a potential selection bias due to possible increased interest in the survey by those who have had positive experiences with the subject of the survey. In order to assess the magnitude of this bias, we picked 10 trainees at random from those who had not responded to the survey. We called these 10 trainees and asked for their reasons for not responding. The 10 gave the following answers:

- 1 indicated that he had not been able to apply the material.
- 1 indicated that he had not followed the training and was on the list of trainees by mistake.
- 8 indicated that they had been too busy to respond to such an extensive survey.

This seems to indicate that the proportion of trainees who had not been able to apply any of the material is roughly the same for those who responded to the survey as those who did not respond, implying there is no significant selection bias.

Another threat is in the survey population: all results are subject to the perception of the architects. A good example is the architects' subjective evaluation of the impact of the practices. The post survey expert workshop noticed that practices that reenforce the importance of the architects and their skills tend to get higher impact ratings. Examples of this phenomenon are:

- The highest rated impact is for the Solution Shaping Workshop, which puts the solution architect in a key position right at the beginning of the solution shaping process.

- Formal evaluation practices like se.ev and ss.ev are sometimes seen as reducing the architect's importance, since they require the architect to justify their decisions; they get a relatively low impact rating.

- Architectural Requirements Prioritization directs the architect in his priorities - and gets a much lower impact rating than the related Documenting Architectural Decisions, which positions the architect as an ("important") decision maker.

The only way to assess the seriousness of this bias is to measure the impact of the practices in ways that exclude the architect's opinion.

Like with the other surveys in this thesis, the results are influenced by cultural aspects of both the Logica company and the Netherlands location, and should be used with care when applied outside of these boundaries.

10 Concluding Remarks

In this chapter, we present our key conclusions. We revisit the research questions and summarize how they have been answered in the thesis, and list the novel contributions to the field. We also discuss promising directions for further research on the topics treated.

10.1 Conclusions

The key high-level conclusions of this thesis are listed below.

On the topic of solution architecting:

1. *Solution architecting is a risk- and cost management discipline.* This is demonstrated in Chapters 8 and 9. The Risk- and Cost Driven solution architecting approach has significant positive impact on the work of most solution architects trained in it.

2. *CMMI support for architecting has improved significantly with version 1.3, but could still be further improved by adding guidance for architecture governance and architecting during the sales phase.* See Chapter 6.

3. *Dealing with emotions is a crucial factor in how architectural knowledge sharing leads to successful projects.* See Chapter 7.

On the topic of NFRs:

4. *Critical NFRs should be quantified, but we should beware of* premature *quantification.* See Chapter 3.

5. *Tendering rules and regulations have a detrimental effect on the quality of IT solutions. The key to successful IT solutions is in trust between customers and suppliers.* Also from Chapter 3.

6. *Modifiability deserves more attention than it is getting now.* Observed in Chapter 4.

One final overall conclusion:

7. *Good solution architecting is not so much a technical problem, but rather a socioeconomic one.* The most important observations above are not technical in nature. They revolve around non-technical keywords like trust, emotions, risk and cost, responsibility and authority.

In the remainder of this chapter, we will show how these conclusions are underpinned by the material presented in this thesis.

10.2 Contributions

Our journey towards improving solution architecting practices in Logica gave us the opportunity to research a number of interesting questions, presented in the introduction to this thesis (Chapter 1). By looking at how these questions were answered, we will now summarize our new contributions to the field, and relate them to the conclusions presented above.

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

How can a solution be structured to best address conflicting Non-Functional Requirements? (RQ-1a)

In Chapter 2, we present a new framework that both provides a model and a repeatable method to transform conflicting requirements into a system decomposition, called Non-Functional Decomposition. NFD is a technique, based on the relationship between functional and non-functional requirements, that brings more clarity and structure in the mapping of requirements onto a solution architecture. Our new framework reveals rationale behind existing architectural patterns and tactics, and can be helpful in developing new patterns and tactics to deal with conflicting NFRs.

What is the best way to quantify Non-Functional Requirements across a contractual divide between customer and supplier? (RQ-1b)

In Chapter 3, we identify some key issues related to NFR quantification in customer/supplier relationships. We argue that economic justification of NFR quantification requires knowledge of the solution architecture.

We argue that critical NFRs should be quantified, but we should beware of *premature* quantification: as our real-life examples illustrate, prematurely quantified NFRs can cripple projects and lead to diverging points of view in customer/supplier relationships that are very hard to resolve. Optimal quantification requires sharing of information between customer and supplier, and it requires time to establish at least a reasonably proven estimate for the cost and value relationships. We suggest a possible way to create better NFR quantification circumstances for customers and suppliers: by means of a requirements convergence plan.

We conclude that trust between customers and suppliers in the IT industry is key to successful solutions. This is a matter of attitude. With the ever growing complexity of IT systems and projects, transparency and awareness between customers and suppliers about NFRs is essential to the feasibility of IT projects. So is willingness to share the risk of unquantified NFRs. Both transparency and risk sharing cannot exist without trust.

How do architects perceive and deal with non-functional requirements? (RQ-1c)

Chapter 4 presents the results 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. IT projects where modifiability is relatively business critical perform significantly worse on average. Our conclusion is that modifiability deserves more attention than it is getting now, especially because in general it is quantified and verified considerably less than other NFRs. Practitioners should be careful when dealing with IT projects with a strong focus on modifiability. We advise to pay particular attention to aspects like managing customer expectations, because it seems that customer satisfaction especially is significantly lower on average in this type of IT projects.

Furthermore, IT projects that applied NFR verification techniques relatively early in development were more successful on average than IT projects that did not apply verification techniques (or applied it relatively late in development). Thus, practitioners

should be aware that the long term benefits of verification outweigh the short term extra costs.

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

What is the nature of solution architecting in the business context of a large IT services company? (RQ-2a)

In Chapter 8, we present our innovative view on the nature of solution architecture: as a risk- and cost management discipline. This view extends existing views of architecture as a higher level abstraction and as a set of design decisions. In comparison with these existing views, it helps architects better order their work, and it helps in better communicating about the architecture with stakeholders in business terms. We also provide guidance on implementing this view in industrial contexts.

What requirements does an architecture process need to fulfill in order to comply with CMMI maturity level 3? (RQ-2b)

Chapter 6 identifies the requirements to make a generic architecting process compliant with CMMI Maturity Level 3, and analyzes the process areas significant to architecting. Our conclusions are that architecture is not a well-defined concept in the CMMI 1.1, but it is improved in later versions of CMMI. CMMI can still be improved in the areas of architecture governance, facilitating the sales phase and learning from architectural choices.

How do architectural knowledge sharing practices relate to challenges in solution delivery projects? (RQ-2c)

In Chapter 7, we describe a survey to gain insight into the mechanisms around architectural knowledge sharing in projects. The analysis shows that architects face many challenges sharing architectural knowledge in projects, especially in large projects. Most of the common challenges appear to be generally neutralized somehow, since they show no correlation with project success. The only challenges that *are* correlated with project success are the ones related to interpersonal relationships. We conclude that *dealing with emotions* is a crucial factor in how architectural knowledge sharing leads to successful projects.

What architecting practices address an IT service provider's business requirements, and what guidance should they contain? (RQ-2d)

Logica's Risk- and Cost Driven Architecture approach is presented in Chapter 9. By combining ideas from [Jacobson et al., 2007] and [Kazman et al., 2006], we document architecting component techniques in practices, embedded in a new framework of structural dimensions to ease identification and integration of best fit practices in a particular context. The practices were harvested from Logica practitioners and enhanced by research. In a way, RCDA is the logical result and culmination of the work presented in this thesis:

- In Part I, we researched ways to handle NFRs; the resulting guidance is embedded in the RCDA practices *Dealing with Non-Functional Requirements* and *Requirements Convergence Planning*.
- In Chapter 5, we saw the importance of an environment where architects can argue their choices and priorities in an objective manner, and select practices that best fit those priorities, rather than follow fashion. RCDA stimulates such an environment by introducing practices that objectify architectural decisions and priorities, and put them in a business context.
- The RCDA core practices constitute a solution architecting process that fulfills the requirements of CMMI Maturity Level 3, as analyzed in Chapter 6.
- RCDA helps architects to deal with emotions (Chapter 7) by smoothing their communication with their solutions' stakeholders; translating architectural concerns and decisions into business terms like risk and cost (Chapter 8).

What is the effect of training architects in such architecting practices? (RQ-2e)

The results of a survey among architects trained in RCDA indicate that for the majority of trainees, the approach has significant positive impact on their solution architecting work. This is true for RCDA as a whole, for its principles, and for its individual practices. The positive effects, however, appear to be much stronger if the architects are in a position of responsibility and authority.

10.3 Discussion

The research presented in this thesis started out with the goal of finding out how to architect IT solutions that adequately serve their purpose, especially in client/supplier

situations. The research was performed within the context of technical assurance: assuring the feasibility, suitability and acceptability of solutions. Although the work treated many technical aspects, the most important conclusions are not technical in nature. They revolve around non-technical keywords like trust (Chapter 3), emotions (Chapter 7), risk and cost (Chapter 8), responsibility and authority (Chapter 9). Perhaps the main conclusion of this thesis should be that the problem of good solution architecting is not so much a technical problem, but rather a socioeconomic one; a conclusion already hinted at in e.g. [Clerc et al., 2007, Sutcliffe, 2008].

10.3.1 Future directions

Within Logica, the journey to improve solution architecting practices will continue. RCDA will be extended with new practices, some of which have already been identified in Chapter 9. More and more architects will be trained and gain experience applying the guidance. As evidence of the benefits of RCDA grows, parts of the approach will get an increasingly formal status in the company's business management system. The success of RCDA has prompted other engineering disciplines within the company to structure their guidance according to the Jacobson-like practices approach [Jacobson et al., 2007]. All of this will lead to new opportunities for research. One promising direction for such research is to relate the application of solution architecting principles and practices to actual *metrics* gathered in projects, rather than depend on surveys among architects. Another obvious extension of the research presented here is to repeat the research outside of Logica, and outside of the Netherlands.

Another interesting direction for research would be the relationship between solution architecting practices and architecting maturity. Could the identification of the solution architecting practices help in assessing an organization's solution architecting maturity, e.g. by enhancing or replacing existing Architecture Maturity Models like the IT Architecture Capability Maturity Model (ACMM) [US Department of Commerce, 2007]? Or could they be used to improve the assessment of individual architects' level of competence in frameworks like Open CA (formerly ITAC) [The Open Group]?

In this thesis, we have taken insights gathered in the software architecture community, and successfully applied them to the wider area of solution architecture. An interesting avenue of exploration would be to find out how the solution architecture principles and practices discussed here relate to other architecture genres, such as systems architecture and enterprise architecture. [Clements, 2009] provides a good basis for such work.

Samenvatting

Naarmate Informatie- en Communicatietechologie een steeds belangrijkere plaats in ons leven inneemt, groeit ook de invloed van de ontwerpbeslissingen die de ICT-oplossingen hun vorm geven. Wij voelen deze invloed in onze verwondering over de nieuwe mogelijkheden die ontwikkelingen als het Internet brengen - mogelijkheden die onze beleving van maatschappelijke interactie binnen één generatie ingrijpend veranderd hebben. Maar de uitwerking van ICT-ontwerpbeslissingen is niet altijd positief. Vaak voelen we negatieve invloed als kleine irritaties, zoals onze kinderen die klagen wanneer hun favoriete social media site kleine veranderingen doorvoert. Soms gaat het echter écht mis, met verreikende gevolgen.

Van tijd tot tijd doet een enkele foutieve beslissing haar invloed gelden in het politieke landschap van een heel land, of zelfs wereldwijd. In de afgelopen 10 jaar kende Nederland een aantal van dit soort mislukkingen, zoals het slecht werkende C2000 communicatiesysteem voor noodhulpverleners, de fraudegevoelige OV-chipkaart en de jarenlange vertraging in het openstellen van tunnels vanwege kwaliteitsproblemen in de veiligheidssoftware. In al deze drie gevallen lag het probleem niet in de functionaliteit van de oplossingen: het waren de andere, “extra-functionele” aspecten waar het misging. Deze extra-functionele aspecten worden in het Engels vaak aangeduid als non-functional requirements (NFRs). Het gaat hier om kwaliteitsattributen als prestatie, vertrouwelijkheid, veiligheid en betrouwbaarheid. De laatste tientallen jaren is steeds duidelijker geworden dat deze attributen worden bepaald door de architectuur van de oplossing, en dat de vereisten aan deze kwaliteitsattributen dus leidend zouden moeten zijn bij het ontwerpen van de oplossingsarchitectuur.

Een rapport van Dalcher en Genus uit 2003 schat de totale financiële kosten van falende ICT-projecten in de VS en de EU op 290 miljard dollar per jaar. Wellicht belangrijker is de significante invloed die bovenstaande voorbeelden hebben op onze kwaliteit van leven. Sommige zijn zelfs levensbedreigend. Om deze problemen op te kunnen lossen is er een beter begrip van extra-functionele aspecten en hun uitwerking in de architectuur van ICT-oplossingen nodig.

Dit proefschrift is tot stand gekomen in de context van een verbetertraject voor architectuurpraktijken in Logica, een grote Europese ICT-dienstverlener. Dit traject begon in 2003, toen we vaststelden dat er behoefte was aan een beter begrip van de invloed van extra-functionele aspecten op onze oplossingen. Het eerste deel van dit proefschrift richt zich op het onderzoek naar deze aspecten.

De eerste vraag die wordt behandeld is hoe oplossingen zodanig gestructureerd kunnen worden dat extra-functionele aspecten optimaal geaddresseerd worden. Ons onderzoek naar deze vraag heeft geleid tot “Non-Functional Decomposition” (NFD), een nieuw raamwerk waarmee conflicterende systeemeisen kunnen worden gebruikt

om een optimale oplossingsstructuur te bepalen. NFD is gebaseerd op de relatie tussen functionele en extra-functionele systeemeisen, en verduidelijkt de samenhang tussen eisen aan en architectuur van een oplossing. Ons raamwerk werpt nieuw licht op de redenen waarom bestaande architectuurpatronen werken, en kan helpen bij het ontwikkelen van nieuwe patronen om conflicterende NFRs aan te pakken.

Een gemeenschappelijke factor in veel problematische ICT-projecten is de traditionele opdrachtgever/leverancier situatie, waarbij de eisen waaraan de oplossing moet voldoen (de "wat"-vraag) wordt opgesteld door een opdrachtgever en het architectuurontwerp (de "hoe"-vraag) door één of meer leveranciers. Een sleutelprobleem in deze situaties is het kwantificeren van de NFRs: het bepalen van de vereiste getalswaarde waaraan een kwaliteitsattribuut van een oplossing moet voldoen. Uit ons onderzoek blijkt dat een optimale kwantificatie van NFRs aanmerkelijk wordt gehinderd door de beperkingen die aanbestedingsregels opleggen aan de communicatie tussen leverancier en opdrachtgever. De aanbestedingsregels kunnen een opdrachtgever zelfs dwingen om de leverancier te kiezen die het *slechtste* begrip heeft van de impact van de extra-functionele eisen. Het blijkt uit economisch oogpunt verstandiger om het kwantificeren van NFRs uit te stellen totdat opdrachtgever en leverancier voldoende tijd hebben gehad om de waarde en kosten die met deze eisen samenhangen in kaart te brengen en met elkaar te delen. Ons onderzoek wijst in de richting van een aantal mogelijke oplossingen, waaronder het breder inzetten van de concurrentiegerichte dialoog bij publieke aanbestedingen van ICT-projecten. Het kernprobleem oplossen vergt echter een verandering in houding: beide partijen dienen het vertrouwen te hebben om informatie over de impact van extra-functionele aspecten aan hun zijde van het contract te delen, en een gelijkwaardig aandeel in de betreffende risico's te aanvaarden.

Uit een enquete onder architecten blijkt dat de meeste extra-functionele eisen geen meetbare negatieve invloed hebben op het succes van ICT-projecten, zolang de architect zich tenminste bewust is van die eisen. De uitzondering op deze regel is modificeerbaarheid: projecten waarin dit aspect van kritisch belang wordt geacht presteren gemiddeld significant slechter dan andere projecten. Onze conclusie is dat modificeerbaarheid meer aandacht verdient dan het nu krijgt, vooral omdat het over het algemeen minder wordt gekwantificeerd en geverifieerd dan andere NFRs. Architecten dienen zorgvuldig te handelen in projecten met een sterke focus op modificeerbaarheid. Aangezien volgens ons onderzoek met name de klanttevredenheid gemiddeld lager is in dit type projecten, adviseren wij om vooral aandacht te schenken aan de verwachtingen rondom de modificeerbaarheid van de oplossing.

Het hierboven genoemde verbetertraject voor architectuurpraktijken binnen Logica kreeg in 2006 meer vaart en richting, toen de Technical Board van het bedrijf de noodzaak van een standaard architectuur-aanpak uitte. Het uiteindelijke resultaat was een nieuwe aanpak voor oplossingsarchitectuur: Risk- and Cost-Driven Architec-

ture (RCDA). De voedingsbodem voor zowel het onderzoek in dit proefschrift als de RCDA aanpak was de functie van "Technical Assurance", een technisch-geweten rol die de auteur vanaf 2005 bekleedde binnen het bedrijf. Vanuit die rol was er intensieve interactie met honderden ICT-projecten van uiteenlopende omvang en complexiteit, en dat is één van de belangrijkste bronnen voor het onderzoek dat hier gepresenteerd wordt. Een andere belangrijke bron is de internationale architectengemeenschap binnen het bedrijf. Dit alles heeft geleid tot het tweede deel van dit proefscrhift, dat zich richt op het vinden van een goede oplossingsarchitectuur-aanpak, en eindigt met het presenteren van RCDA.

De grondslag voor RCDA ligt in nieuwe inzichten in de aard van oplossingsarchitectuur. In de loop der jaren zijn wij architectuur gaan zien als een discipline om risico's en kosten te beheersen. Deze visie is een extensie van bestaande visies op architectuur als abstractieniveau en als een verzameling ontwerpbeslissingen. In vergelijking met deze bestaande visies helpt de risico- en kostengedreven aanpak architecten bij het ordenen van hun werk, en bij het communiceren erover met belanghebbenden in zakelijke termen. RCDA is een verzameling praktijken, ingebed in een raamwerk dat het selecteren en toepassen ervan in de praktijk faciliteert. De praktijken zijn geoogst vanuit het bedrijf en de literatuur, en aangescherpt door onderzoek. Een enquete onder de in RCDA getrainde architecten wijst uit dat de aanpak voor de meerderheid van hen een significant positieve invloed heeft op hun werk. Dit geldt zowel voor de RCDA-aanpak, als voor de principes erachter en de individuele praktijken die de aanpak omvat. Tevens blijkt dat de aanpak verreweg het beste werkt als de architect een positie met verantwoordelijkheid en autoriteit bekleedt, bijvoorbeeld als lead-architect.

Bij het tot stand komen van RCDA hebben we naar twee aspecten nog afzonderlijk onderzoek verricht: de eisen die procesverbeterstandaard CMMI aan een architectuuraanpak stelt, en de rol van architectuurkennisdeling in ICT-projecten. Uit het onderzoek blijkt dat de nieuwste versie van CMMI, 1.3, aanmerkelijk beter is ten opzichte van versie 1.1 in het ondersteunen van architectuurprocessen. De gebieden waarop nog meer verbetering kan worden bereikt zijn het managen van aan architectuur-gerelateerde hulpbronnen, het bedrijven van architectuur in de verkoopfase en het leren van architectuurkeuzes. Voor het onderzoek naar de rol van architectuurkennisdeling hebben we opnieuw een enquete onder architecten uitgevoerd. Uit de analyse van de enqueteresultaten blijkt dat architecten veel uitdagingen moeten overwinnen bij het delen van architectuurkennis in projecten, vooral in grotere projecten. De meest voorkomende uitdagingen tonen geen correlatie met projectsucces, dus ze worden op de een of andere manier over het algemeen geneutraliseerd. De uitdagingen die wél gecorrelleerd zijn met projectsucces hebben allemaal iets te maken met intermenselijke relaties. Wij concluderen dat het *omgaan met emoties* een cruciale factor is in hoe architectuurkennisdeling leidt tot succesvolle projecten.

Uit dit alles kan nog één overkoepelende conclusie getrokken worden. Oplossingsarchitectuur is een uit de Informatie- en Communicatie*technologie* voortgekomen discipline. De belangrijkste bevindingen uit ons onderzoek wijzen echter hoofdzakelijk op het belang van niet-technische begrippen als vertrouwen, emoties, risico's en kosten, verantwoordelijkheid en autoriteit. Onze eindconclusie is dan ook dat goed oplossingsarchitectuur bedrijven niet zozeer een technische, alswel een socio-economische uitdaging is.

SIKS Dissertation Series

2009-01 Rasa Jurgelenaite (RUN)
Symmetric Causal Independence Models
2009-02 Willem Robert van Hage (VU)
Evaluating Ontology-Alignment Techniques
2009-03 Hans Stol (UvT)
A Framework for Evidence-based Policy Making Using IT
2009-04 Josephine Nabukenya (RUN)
Improving the Quality of Organisational Policy Making using Collaboration Engineering
2009-05 Sietse Overbeek (RUN)
Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality
2009-06 Muhammad Subianto (UU)
Understanding Classification
2009-07 Ronald Poppe (UT)
Discriminative Vision-Based Recovery and Recognition of Human Motion
2009-08 Volker Nannen (VU)
Evolutionary Agent-Based Policy Analysis in Dynamic Environments
2009-09 Benjamin Kanagwa (RUN)
Design, Discovery and Construction of Service-oriented Systems
2009-10 Jan Wielemaker (UVA)
Logic programming for knowledge-intensive interactive applications
2009-11 Alexander Boer (UVA)
Legal Theory, Sources of Law & the Semantic Web
2009-12 Peter Massuthe (TUE, Humboldt-Universitaet zu Berlin)
Operating Guidelines for Services
2009-13 Steven de Jong (UM)
Fairness in Multi-Agent Systems
2009-14 Maksym Korotkiy (VU)
From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)
2009-15 Rinke Hoekstra (UVA)
Ontology Representation - Design Patterns and Ontologies that Make Sense
2009-16 Fritz Reul (UvT)
New Architectures in Computer Chess
2009-17 Laurens van der Maaten (UvT)
Feature Extraction from Visual Data
2009-18 Fabian Groffen (CWI)
Armada, An Evolving Database System
2009-19 Valentin Robu (CWI)
Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets
2009-20 Bob van der Vecht (UU)
Adjustable Autonomy: Controling Influences on Decision Making
2009-21 Stijn Vanderlooy (UM)
Ranking and Reliable Classification
2009-22 Pavel Serdyukov (UT)
Search For Expertise: Going beyond direct evidence
2009-23 Peter Hofgesang (VU)
Modelling Web Usage in a Changing Environment
2009-24 Annerieke Heuvelink (VUA)
Cognitive Models for Training Simulations
2009-25 Alex van Ballegooij (CWI)
”RAM: Array Database Management through Relational Mapping”
2009-26 Fernando Koch (UU)
An Agent-Based Model for the Development of Intelligent Mobile Services
2009-27 Christian Glahn (OU)
Contextual Support of social Engagement and Reflection on the Web

2009-28 Sander Evers (UT)
Sensor Data Management with Probabilistic Models
2009-29 Stanislav Pokraev (UT)
Model-Driven Semantic Integration of Service-Oriented Applications
2009-30 Marcin Zukowski (CWI)
Balancing vectorized query execution with bandwidth-optimized storage
2009-31 Sofiya Katrenko (UVA)
A Closer Look at Learning Relations from Text
2009-32 Rik Farenhorst (VU) and Remco de Boer (VU)
Architectural Knowledge Management: Supporting Architects and Auditors
2009-33 Khiet Truong (UT)
How Does Real Affect Affect Affect Recognition In Speech?
2009-34 Inge van de Weerd (UU)
Advancing in Software Product Management: An Incremental Method Engineering Approach
2009-35 Wouter Koelewijn (UL)
Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling
2009-36 Marco Kalz (OUN)
Placement Support for Learners in Learning Networks
2009-37 Hendrik Drachsler (OUN)
Navigation Support for Learners in Informal Learning Networks
2009-38 Riina Vuorikari (OU)
Tags and self-organisation: a metadata ecology for learning resources in a multilingual context
2009-39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin)
Service Substitution – A Behavioral Approach Based on Petri Nets
2009-40 Stephan Raaijmakers (UvT)
Multinomial Language Learning: Investigations into the Geometry of Language
2009-41 Igor Berezhnyy (UvT)
Digital Analysis of Paintings
2009-42 Toine Bogers
Recommender Systems for Social Bookmarking
2009-43 Virginia Nunes Leal Franqueira (UT)
Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients
2009-44 Roberto Santana Tapia (UT)
Assessing Business-IT Alignment in Networked Organizations
2009-45 Jilles Vreeken (UU)
Making Pattern Mining Useful
2009-46 Loredana Afanasiev (UvA)
Querying XML: Benchmarks and Recursion
2010-01 Matthijs van Leeuwen (UU)
Patterns that Matter
2010-02 Ingo Wassink (UT)
Work flows in Life Science
2010-03 Joost Geurts (CWI)
A Document Engineering Model and Processing Framework for Multimedia documents
2010-04 Olga Kulyk (UT)
Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments
2010-05 Claudia Hauff (UT)
Predicting the Effectiveness of Queries and Retrieval Systems
2010-06 Sander Bakkes (UvT)
Rapid Adaptation of Video Game AI
2010-07 Wim Fikkert (UT)
Gesture interaction at a Distance
2010-08 Krzysztof Siewicz (UL)
Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments
2010-09 Hugo Kielman (UL)
A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging

2010-10 Rebecca Ong (UL)
Mobile Communication and Protection of Children 2010-11 Adriaan Ter Mors (TUD)
The world according to MARP: Multi-Agent Route Planning
2010-12 Susan van den Braak (UU)
Sensemaking software for crime analysis
2010-13 Gianluigi Folino (RUN)
High Performance Data Mining using Bio-inspired techniques
2010-14 Sander van Splunter (VU)
Automated Web Service Reconfiguration
2010-15 Lianne Bodenstaff (UT)
Managing Dependency Relations in Inter-Organizational Models
2010-16 Sicco Verwer (TUD)
Efficient Identification of Timed Automata, theory and practice
2010-17 Spyros Kotoulas (VU)
Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications
2010-18 Charlotte Gerritsen (VU)
Caught in the Act: Investigating Crime by Agent-Based Simulation
2010-19 Henriette Cramer (UvA)
People's Responses to Autonomous and Adaptive Systems
2010-20 Ivo Swartjes (UT)
Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative
2010-21 Harold van Heerde (UT)
Privacy-aware data management by means of data degradation
2010-22 Michiel Hildebrand (CWI)
End-user Support for Access to Heterogeneous Linked Data
2010-23 Bas Steunebrink (UU)
The Logical Structure of Emotions
2010-24 Dmytro Tykhonov
Designing Generic and Efficient Negotiation Strategies
2010-25 Zulfiqar Ali Memon (VU)
Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective
2010-26 Ying Zhang (CWI)
XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines
2010-27 Marten Voulon (UL)
Automatisch contracteren
2010-28 Arne Koopman (UU)
Characteristic Relational Patterns
2010-29 Stratos Idreos(CWI)
Database Cracking: Towards Auto-tuning Database Kernels
2010-30 Marieke van Erp (UvT)
Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval
2010-31 Victor de Boer (UVA)
Ontology Enrichment from Heterogeneous Sources on the Web
2010-32 Marcel Hiel (UvT)
An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems
2010-33 Robin Aly (UT)
Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval
2010-34 Teduh Dirgahayu (UT)
Interaction Design in Service Compositions
2010-35 Dolf Trieschnigg (UT)
Proof of Concept: Concept-based Biomedical Information Retrieval
2010-36 Jose Janssen (OU)
Paving the Way for Lifelong Learning; Facilitating competence development through a learning path specification
2010-37 Niels Lohmann (TUE)
Correctness of services and their composition
2010-38 Dirk Fahland (TUE)
From Scenarios to components

2010-39 Ghazanfar Farooq Siddiqui (VU)
Integrative modeling of emotions in virtual agents
2010-40 Mark van Assem (VU)
Converting and Integrating Vocabularies for the Semantic Web
2010-41 Guillaume Chaslot (UM)
Monte-Carlo Tree Search
2010-42 Sybren de Kinderen (VU)
Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach
2010-43 Peter van Kranenburg (UU)
A Computational Approach to Content-Based Retrieval of Folk Song Melodies
2010-44 Pieter Bellekens (TUE)
An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain
2010-45 Vasilios Andrikopoulos (UvT)
A theory and model for the evolution of software services
2010-46 Vincent Pijpers (VU)
e3alignment: Exploring Inter-Organizational Business-ICT Alignment
2010-47 Chen Li (UT)
Mining Process Model Variants: Challenges, Techniques, Examples
2010-48 Milan Lovric (EUR)
Behavioral Finance and Agent-Based Artificial Markets
2010-49 Jahn-Takeshi Saito (UM)
Solving difficult game positions
2010-50 Bouke Huurnink (UVA)
Search in Audiovisual Broadcast Archives
2010-51 Alia Khairia Amin (CWI)
Understanding and supporting information seeking tasks in multiple sources
2010-52 Peter-Paul van Maanen (VU)
Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention
2010-53 Edgar Meij (UVA)
Combining Concepts and Language Models for Information Access
2011-01 Botond Cseke (RUN)
Variational Algorithms for Bayesian Inference in Latent Gaussian Models
2011-02 Nick Tinnemeier(UU)
Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language
2011-03 Jan Martijn van der Werf (TUE)
Compositional Design and Verification of Component-Based Information Systems
2011-04 Hado van Hasselt (UU)
Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference learning algorithms
2011-05 Base van der Raadt (VU)
Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.
2011-06 Yiwen Wang (TUE)
Semantically-Enhanced Recommendations in Cultural Heritage
2011-07 Yujia Cao (UT)
Multimodal Information Presentation for High Load Human Computer Interaction
2011-08 Nieske Vergunst (UU)
BDI-based Generation of Robust Task-Oriented Dialogues
2011-09 Tim de Jong (OU)
Contextualised Mobile Media for Learning
2011-10 Bart Bogaert (UvT)
Cloud Content Contention
2011-11 Dhaval Vyas (UT)
Designing for Awareness: An Experience-focused HCI Perspective
2011-12 Carmen Bratosin (TUE)
Grid Architecture for Distributed Process Mining

2011-13 Xiaoyu Mao (UvT)
Airport under Control. Multiagent Scheduling for Airport Ground Handling
2011-14 Milan Lovric (EUR)
Behavioral Finance and Agent-Based Artificial Markets
2011-15 Marijn Koolen (UvA)
The Meaning of Structure: the Value of Link Evidence for Information Retrieval
2011-16 Maarten Schadd (UM)
Selective Search in Games of Different Complexity
2011-17 Jiyin He (UVA)
Exploring Topic Structure: Coherence, Diversity and Relatedness
2011-18 Mark Ponsen (UM)
Strategic Decision-Making in complex games
2011-19 Ellen Rusman (OU)
The Mind ' s Eye on Personal Profiles
2011-20 Qing Gu (VU)
Guiding service-oriented software engineering - A view-based approach
2011-21 Linda Terlouw (TUD)
Modularization and Specification of Service-Oriented Systems
2011-22 Junte Zhang (UVA)
System Evaluation of Archival Description and Access
2011-23 Wouter Weerkamp (UVA)
Finding People and their Utterances in Social Media
2011-24 Herwin van Welbergen (UT)
Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior
2011-25 Syed Waqar ul Qounain Jaffry (VU)
Analysis and Validation of Models for Trust Dynamics
2011-26 Matthijs Aart Pontier (VU)
Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots
2011-27 Aniel Bhulai (VU)
Dynamic website optimization through autonomous management of design patterns
2011-28 Rianne Kaptein(UVA)
Effective Focused Retrieval by Exploiting Query Context and Document Structure
2011-29 Faisal Kamiran (TUE)
Discrimination-aware Classification
2011-30 Egon van den Broek (UT)
Affective Signal Processing (ASP): Unraveling the mystery of emotions
2011-31 Ludo Waltman (EUR)
Computational and Game-Theoretic Approaches for Modeling Bounded Rationality
2011-32 Nees-Jan van Eck (EUR)
Methodological Advances in Bibliometric Mapping of Science
2011-33 Tom van der Weide (UU)
Arguing to Motivate Decisions
2011-34 Paolo Turrini (UU)
Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations
2011-35 Maaike Harbers (UU)
Explaining Agent Behavior in Virtual Training
2011-36 Erik van der Spek (UU)
Experiments in serious game design: a cognitive approach
2011-37 Adriana Burlutiu (RUN)
Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference
2011-38 Nyree Lemmens (UM)
Bee-inspired Distributed Optimization
2011-39 Joost Westra (UU)
Organizing Adaptation using Agents in Serious Games
2011-40 Viktor Clerc (VU)
Architectural Knowledge Management in Global Software Development

2011-41 Luan Ibraimi (UT)
Cryptographically Enforced Distributed Data Access Control
2011-42 Michal Sindlar (UU)
Explaining Behavior through Mental State Attribution
2011-43 Henk van der Schuur (UU)
Process Improvement through Software Operation Knowledge
2011-44 Boris Reuderink (UT)
Robust Brain-Computer Interfaces
2011-45 Herman Stehouwer (UvT)
Statistical Language Models for Alternative Sequence Selection
2011-46 Beibei Hu (TUD)
Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work
2011-47 Azizi Bin Ab Aziz(VU)
Exploring Computational Models for Intelligent Support of Persons with Depression
2011-48 Mark Ter Maat (UT)
Response Selection and Turn-taking for a Sensitive Artificial Listening Agent
2011-49 Andreea Niculescu (UT)
Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality
2012-01 Terry Kakeeto (UvT)
Relationship Marketing for SMEs in Uganda
2012-02 Muhammad Umair(VU)
Adaptivity, emotion, and Rationality in Human and Ambient Agent Models
2012-03 Adam Vanya (VU)
Supporting Architecture Evolution by Mining Software Repositories
2012-04 Jurriaan Souer (UU)
Development of Content Management System-based Web Applications
2012-05 Marijn Plomp (UU)
Maturing Interorganisational Information Systems
2012-06 Wolfgang Reinhardt (OU)
Awareness Support for Knowledge Workers in Research Networks
2012-07 Rianne van Lambalgen (VU)
When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions
2012-08 Gerben de Vries (UVA)
Kernel Methods for Vessel Traject
2012-09 Ricardo Neisse (UT)
Trust and Privacy Management Support for Context-Aware Service Platforms
2012-10 David Smits (TUE)
Towards a Generic Distributed Adaptive Hypermedia Environment
2012-11 J.C.B. Rantham Prabhakara (TUE)
Process Mining in the Large: Preprocessing, Discovery, and Diagnostics
2012-12 Kees van der Sluijs (TUE)
Model Driven Design and Data Integration in Semantic Web Information Systems
2012-13 Suleman Shahid (UvT)
Fun and Face: Exploring non-verbal expressions of emotion during playful interactions
2012-14 Evgeny Knutov(TUE)
Generic Adaptation Framework for Unifying Adaptive Web-based Systems
2012-15 Natalie van der Wal (VU)
Intelligent Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes
2012-16 Fiemke Both (VU)
Helping people by understanding them - Ambient Agents supporting task execution and depression treatment
2012-17 Amal Elgammal (UVT)
Towards a Comprehensive Framework for Business Process Compliance
2012-18 Eltjo R. Poort (VU)
Improving Solution Architecting Practices

Bibliography

AACE. Risk analysis and contingency determination using range estimating, 2000. AACE International Recommended Practice No. 41R-08. 128

G. Abowd, L. Bass, P. Clements, R. Kazman, L. Northrop, and A. Zaremski. Recommended best industrial practice for software architecture evaluation. Technical Report CMU/SEI-96-TR-025, SEI, 1997. 149

P. Abrahamsson, M. A. Babar, and P. Kruchten. Agility and architecture: Can they coexist? *IEEE Software*, 27:16–22, 2010. 142

Agile Alliance. Manifesto for agile software development, 2001. URL `http://agilemanifesto.org`. 52, 141

M. Ali Babar, T. Dingsøyr, P. Lago, and H. van Vliet, editors. *Software Architecture Knowledge Management: Theory and Practice*. Springer, Aug. 2009. 10, 101, 117

D. Baccarini. The logical framework method for defining project success. *Project Management Journal*, 30:25–32, 1999. 61

S. F. Bacon. *Religious Meditations*. 1597. 107

P. Bannerman. Risk and risk management in software projects: A reassessment. *Journal of Systems and Software*, 81:2118–2133, 2008. 139

M. R. Barbacci, R. Ellison, A. J. Lattanze, J. A. Stafford, C. B. Weinstock, and W. G. Wood. Quality attribute workshops (QAWs), third edition. Technical Report CMY/SEI-2003-TR-016, SEI, 2003. 42, 48, 148

L. Bass and R. Kazman. Architecture based development. Technical Report CMU/SEI-2000-TR-007, SEI, April 1999. 80, 93

L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice, 2nd. ed.* Addison Wesley, 2003. 5, 16, 26, 35, 38, 42, 43, 55, 58, 93, 118, 124, 139, 149

L. Bass, R. Nord, W. Wood, and D. Zubrow. Risk themes discovered through architecture evaluations. In *6th Working IFIP/IEEE Conference on Software Architecture (WICSA)*, pages 1–10. IEEE Computer Society, 2007. 139

R. Berntsson Svensson. *Managing Quality Requirements in Software Product Development*. PhD thesis, Department of Computer Science, Lund University, 2009. 5, 43, 55, 61, 62

S. Biffl, A. Aybuke, B. Boehm, H. Erdogmus, and P. Gruenbacher, editors. *Value-Based Software Engineering*, chapter Valuation of Software Initiatives Under Uncertainty: Concepts, Issues, and Techniques, pages 39–66. Springer, 2006. 135, 140

B. Boehm. *Software Engineering Economics*. Prentice Hall, 1981. 52, 118, 131

B. Boehm. A spiral model of software development and enhancement. *IEEE Computer*, 21(5):61–72, 1988. 139

B. Boehm and P. Bose. A collaborative spiral software process model based on Theory W. In *Third International Conference on the Software Process, 'Applying the Software Process'*, 1994. 16, 38

B. Boehm and H. In. Identifying quality-requirement conflicts. *IEEE Software*, pages 25–35, March 1996. 5, 16, 22, 35, 51

B. Boehm and R. Turner. *Balancing Agility and Discipline*. Addison Wesley, 2004. 139

B. Boehm, P. Bose, E. Horowitz, and M. J. Lee. Software requirements negotiation and renegotiation aids: a Theory W based spiral approach. In *17th international conference on Software engineering (ICSE)*, pages 243 – 253, 1995. 51

B. W. Boehm. Software risk management: Principles and practices. *IEEE Software*, 8: 32–41, 1991. 127, 139

J. Bosch. *Design and Use of Software Architectures*. Addison Wesley, 2000. 16, 35

J. Bosch. Software architecture: The next step. In *Software Architecture, First European Workshop (EWSA)*, volume 3047 of *LNCS*, pages 194–199. Springer, May 2004. 81, 90

F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. John Wiley and Sons Ltd, 1996. 16

CCPSO. Common criteria for information technology security evaluation, August 1999. URL `http://www.commoncriteriaportal.org/cc`. Produced by the Common Criteria Project Sponsoring Organizations, version 2.1. 22, 23

L. Chung, B. Nixon, E. S. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Academic, 1999. 5, 16, 35, 38, 42, 52, 55, 58

P. Clements. Exploring enterprise, system of systems, and system and software architectures, January 2009. URL `http://www.sei.cmu.edu/library/assets/20090122webinar.pdf`. 176

P. Clements and L. Northrop. *Software Product Lines*. Addison Wesley, 2002. 150

P. Clements and M. Shaw. The golden age of software architecture: A comprehensive survey. Technical report, Institute for Software Research International School of Computer Science, Carnegie Mellon University, 2006. CMU-ISRI-06-101. 101

P. Clements, R. Kazman, and M. Klein. *Evaluating Software Architectures*. Addison Wesley, 2002. 42, 78, 81

P. Clements, R. Kazman, M. Klein, D. Devesh, S. Reddy, and P. Verma. The duties, skills, and knowledge of software architects. In *6th Working IFIP/IEEE Conference on Software Architecture (WICSA)*. IEEE Computer Society, 2007. 135, 140, 151

V. Clerc. *Architectural Knowledge Management in Global Software Development*. PhD thesis, VU University Amsterdam, 2011. 117

V. Clerc, P. Lago, and H. van Vliet. The architect's mindset. In *Quality of software architectures 3rd international conference on Software architectures, components, and applications*, QoSA'07, pages 231–249, Berlin, Heidelberg, 2007. Springer-Verlag. 166, 176

CMMI Product Team. CMMI for development, version 1.3. Technical report, SEI, 2010. CMU/SEI-2010-TR-033. 6, 22, 79, 83, 85, 91, 97

H. R. Costa, M. de O. Barros, and G. H. Travassos. Evaluating software project portfolio risks. *Journal of Systems and Software*, 80(1):16–31, 2007. 139

L. J. Cronbach. Coefficient alpha and the internal structure of tests. *Psychometrika*, 16 (3):297–334, 1951. 61, 109

D. Dalcher and A. Genus. Avoiding IS/IT implementation failure. *Technology Analysis and Strategic Management*, 15(4):403–407, December 2003. 2

A. M. Davis. *Just Enough Requirements Management*. Dorset House, 2005. 52

R. Davison, M. G. Martinsons, and N. Kock. Principles of canonical action research. *Information Systems Journal*, 14(1):65–86, 2004. 8

R. de Boer, R. Farenhorst, P. Lago, H. van Vliet, V. Clerc, and A. Jansen. Architectural knowledge: Getting to the core. In S. Overhage, C. Szyperski, R. Reussner, and J. Stafford, editors, *Software Architectures, Components, and Applications*, volume 4880 of *Lecture Notes in Computer Science*, pages 197–214. Springer Berlin / Heidelberg, 2007. 125

H. de Bruin and H. van Vliet. Top-down composition of software architectures. In *9th Annual IEEE International Conference on the Engineering of Computer-Based Systems (ECBS)*, April 2002. 16, 22

B. de Winter. Gratis reizen kan. *PC-Active*, (224):16–18, 2011. 1

D. Dvir, T. Raz, and A. J. Shenhar. An empirical analysis of the relationship between project planning and project success. *International Journal of Project Management*, 21:89–95, 2003. 61

K. E. Emam and A. G. Koru. A replicated survey of IT software project failures. *IEEE Software*, September/October 2008:84–89, 2008. 116

ETSI. European digital cellular telecommunications system (phase 1);technical realization of the short message service point-to-point (GSM 03.40), 1995. URL `http://www.etsi.org`. 75

European Commission. Directive 2004/18/EC of the european parliament and of the council of 31 march 2004 on the coordination of procedures for the award of public works contracts, public supply contracts and public service contracts. Official Journal of the European Union, 2004. 39, 42, 49

Expertgroep C2000. Tussenrapportage expertgroep C2000. Ministerie van Binnenlandse Zaken en Koninkrijksrelaties, 2009. 1

G. Fairbanks. Just enough architecture: The risk-driven model. *Crosstalk*, Nov/Dec 2010. 57, 127, 138, 140, 151

R. Farenhorst and R. de Boer. *Architectural Knowledge Management – Supporting Architects and Auditors*. PhD thesis, VU University Amsterdam, 2009. 3, 101, 117, 121

R. Farenhorst and H. van Vliet. Experiences with a wiki to support architectural knowledge sharing. 2008. 150

M. S. Feather, S. L. Cornford, K. A. Hicks, J. D. Kiper, and T. Menzies. A broad, quantitative model for making early requirements decisions. *IEEE Software*, 25: 49–56, March 2008. 140

M. Fowler. Who needs an architect? *IEEE Software*, 20(5):11–13, 2003. 123, 124, 130, 151

J. Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition*. Addison-Wesley, 1995. 110, 116

S. Fricker and M. Glinz. Comparison of requirements hand-off, analysis, and negotiation: Case study. In *2010 18th IEEE International Requirements Engineering Conference*, RE '10, pages 167–176, Washington, DC, USA, 2010. IEEE Computer Society. 52

S. Fricker, T. Gorschek, C. Byman, and A. Schmidle. Handshaking with implementation proposals: Negotiating requirements understanding. *IEEE Software*, 27(2): 72–80, 2010. 51

E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995. 16, 149

D. A. Garvin. What does "product quality" really mean? *MIT Sloan Management Review*, 26:25–43, Fall 1984. 43

D. L. Gibson, D. Goldenson, and K. Kost. Performance results of CMMI-based process improvement. Technical Report CMU/SEI-2006-TR-004, SEI, August 2006. 83

T. Gilb. *Principles of Software Engineering Management*. Addison Wesley, 1988. 5, 21, 139

T. Gilb. *Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*. Butterworth-Heinemann, Newton, MA, USA, 2005. 5, 21, 24, 42, 53, 82, 141

M. Glinz. On non-functional requirements. In *15th IEEE International Requirements Engineering Conference RE 2007*, pages 21–26. IEEE, 2007. 5

M. Glinz. A risk-based, value-oriented approach to quality requirements. *IEEE Software*, 25:34–41, 2008. 52, 127

R. B. Grady. An economic release decision model: Insights into software project management. In *Applications of Software Measurement Conference, Orange Park, Software Quality Engineering*, pages 227–239, 1999. 62

R. Gram and B. Keulen. Quick scan tunnelprojecten. Ministerie van Verkeer en Waterstaat - Rijkswaterstaat, 2010. Referentie 10 008 R 012. 1, 42

D. Gross and E. Yu. From non-functional requirements to design through patterns. *Requirements Engineering*, 6(1):18–36, 2001. 16

P. Gruenbacher, A. Egyed, and N. Medvidovic. Reconciling software requirements and architecture: the CBSP approach. In *5th IEEE International Symposium on Requirements Engineering*, IEEE CS, August 2001. 16, 35

A. Herrmann and M. Daneva. Requirements prioritization based on benefit and cost prediction: An agenda for future research. In *2008 16th IEEE International Requirements Engineering Conference*, pages 125–134, Washington, DC, USA, 2008. IEEE Computer Society. 142

A. Herrmann and B. Paech. MOQARE: misuse-oriented quality requirements engineering. *Requir. Eng.*, 13:73–86, January 2008. doi: 10.1007/s00766-007-0058-9. 142

A. Herrmann, A. Morali, and S. Etalle. RiskREP: Risk-based security requirements elicitation and prioritization (extended version). Technical Report TR-CTIT-10-28, Centre for Telematics and Information Technology University of Twente, Enschede, August 2010. URL `http://eprints.eemcs.utwente.nl/18342/`. 141

C. Hofmeister, P. Kruchten, R. L. Nord, J. H. Obbink, A. Ran, and P. America. A general model of software architecture design derived from five industrial approaches. *Journal of Systems and Software*, 80(1):106–126, 2007. 80, 94, 132, 133

IEC 61508. Functional safety of electrical/electronic/ programmable electronic safety-related systems, 1999. IEC 61508. 22

ISO 42010. Systems and software engineering - architecture description, 2011. ISO 42010:2011. 4, 96, 123, 124, 125, 149

ISO/IEC 25000. Software engineering – software product quality requirements and evaluation (SQuaRE) – guide to SQuaRE, 2005. ISO 25000:2005. 21, 42

A. Ivanović and P. America. Information needed for architecture decision making. In *2010 ICSE Workshop on Product Line Approaches in Software Engineering*, PLEASE '10, pages 54–57, New York, NY, USA, 2010a. ACM. 122

A. Ivanović and P. America. Strategy-focused architecture decision making. In P. van de Laar and T. Punter, editors, *Views on Evolvability of Embedded Systems*, pages 245–260. Springer, 2010b. 140

M. Jackson. *Problem frames: analyzing and structuring software development problems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001. 35

I. Jacobson, P. W. Ng, and I. Spence. Enough process – let's do practices. *Journal of Object Technology*, 6(6):41–66, July-August 2007. 152, 175, 176

A. Jansen and J. Bosch. Software architecture as a set of architectural design decisions. In *5th Working IEEE/IFIP Conference on Software Architecture*, pages 109–120, Washington, DC, USA, 2005. IEEE Computer Society. 3, 121, 122, 123, 125, 149, 151

E. Johansson, A. Wesslén, L. Bratthall, and M. Höst. The importance of quality requirements in software platform development – a survey. In *HICSS '01: 34th Annual Hawaii International Conference on System Sciences*, volume 9, page 9057, Washington, DC, USA, 2001. IEEE Computer Society. 67

C. Jones. *Software Assessments, Benchmarks, and Best Practices*. Addison-Wesley, 2000. 110, 116

D. Kahneman. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, 2011. 141, 144

D. Kahneman and A. Tversky. Prospect theory: An analysis of decision and risk. *Econometrica*, 47:263–291, 1979. 141

D. Karolak. *Software Engineering Risk Management*. IEEE Computer Society Press,U.S., 1995. 52

R. Kazman, J. Asundi, and M. Klein. Making architecture design decisions: An economic approach. Technical Report CMU/SEI-2002-TR-035, SEI, 2002. 28, 42, 52, 78, 122, 140, 141, 143, 149

R. Kazman, L. Bass, and M. Klein. The essential components of software architecture design and analysis. *Journal of Systems and Software*, 79(8):1207–1216, 2006. 151, 152, 175

A. Klusener, R. Lämmel, and C. Verhoef. Architectural modifications to deployed software. *Science of Computer Programming*, 54:143–211, 2005. 130, 143

N. Kock. *Action Research: Its Nature and Relationship to Human-Computer Interaction*. The Interaction-Design.org Foundation, Aarhus, Denmark, 2011. URL `http://www.interaction-design.org/encyclopedia/action_research.html`. 8

P. Kruchten. The 4+1 view model of architecture. *IEEE Software*, 12(6):45–50, 1995. 149

P. Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley, Boston, 1998. 3, 24, 51, 121, 123

P. Lago and H. van Vliet. Building up and reasoning about architectural knowledge. In *Second International Conference on the Quality of Software Architectures (QoSA)*, pages 43–58, 2006. 101

A. Lamsweerde. Conceptual modeling: Foundations and applications. chapter Reasoning About Alternative Requirements Options, pages 380–397. Springer-Verlag, Berlin, Heidelberg, 2009. 52

P. A. Laplante and C. J. Neill. Uncertainty: A meta-property of software. In *29th Annual IEEE/NASA Software Engineering Workshop*, pages 228–233, 2005. 52

B. Lawrence, K. Wiegers, and C. Ebert. The top risks of requirements engineering. *IEEE Software*, 18(6):62–63, 2001. 5

H. Lee. Applying fuzzy set theory to evaluate the rate of aggregate risk in software development. *Fuzzy Sets and Systems*, 80(3):261–271, 1996. 139

H. K. N. Leung. Quality metrics for intranet applications. *Information and Management*, 38(3):137–152, 2001. 67

K. Linberg. Software developer perceptions about software project failure: a case study. *The Journal of Systems and Software*, 49:177–92, 1999. 61

D. Mairiza, D. Zowghi, and N. Nurmuliani. An investigation into the notion of non-functional requirements. In *2010 ACM Symposium on Applied Computing (SAC), Sierre, Switzerland, March 22-26, 2010*, pages 311–317, 2010. 5, 58

R. Malan and D. Bredemeyer. Less is more with minimalist architecture. *IT Pro*, pages 46–48, September-October 2002. 138, 151

M. L. Markus. Power, politics, and M.I.S. implementation. *Commun. ACM*, 26(6): 430–444, 1983. 118

T. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2: 308–320, 1976. 68

S. McConnell. *Rapid Development*. Microsoft Press, 1996. 118

S. C. Mcconnell. *Software Project Survival Guide (Pro – Best Practices)*. Microsoft Press, Nov. 1997. 52

Money Magazine. Best jobs in america 2010, top 100. CNN on-line, November 2010. URL `http://money.cnn.com/magazines/moneymag/bestjobs/2010/snapshots/1.html`. Rank 1: Software Architect. 136

J. Mylopoulos. Goal-oriented requirements engineering, part ii. In *RE '06: 14th IEEE International Requirements Engineering Conference*, Washington, DC, USA, 2006. IEEE Computer Society. 5, 38, 55

J. Mylopoulos, L. Chung, and B. Nixon. Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Trans. Softw. Eng.*, 18(6):483–497, 1992. 5, 16

J. Noppen. *Imperfect Information in Software Design Processes*. PhD thesis, University of Twente, 2007. 52

H. Obbink, P. Kruchten, W. Kozaczynski, R. Hilliard, A. Ran, H. Postema, L. Dominick, R. Kazman, W. Tracz, and E. Kahane. Report on software architecture review and assessment (SARA). Technical report, SARA Working group, 2002. URL `http://kruchten.com/philippe/architecture/SARAv1.pdf`. retrieved 11 January 2012. 81, 132

B. Paech and D. Kerkow. Non-functional requirements engineering – quality is essential. In *10th Anniversary International Workshop on Requirements Engineering: Foundation for Software Quality*, 2004. 62

B. Paech, R. Heinrich, G. Zorn-Pauli, A. Jung, and S. Tadjiky. Answering a request for proposal – challenges and proposed solutions. In *Proceedings 18th International Working Conference Requirements Engineering: Foundation for Software Quality (REFSQ2012)*. 51

B. Paech, A. Detroit, D. Kerkow, and A. von Knethen. Functional requirements, non-functional requirements, and architecture should not be separated – a position paper. REFSQ, Essen, Germany, September 2002. 5, 38, 52, 55

D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *SIGSOFT Softw. Eng. Notes*, 17:40–52, October 1992. 3, 121

I. Pinkster, B. van de Burgt, D. Janssen, and E. van Veenendaal. *Successful Test Management – An Integral Approach*. Springer, 2004. 42

J. Pinto and D. Slevin. Project success: definitions and measurement techniques. *Project Management Journal*, 19:67–72, 1988. 61

G. Pitette. Progressive acquisition and the RUP: Comparing and combining iterative processes for acquisition and software development. 2001. 51

E. R. Poort and P. H. N. de With. Modelling the relationship between quality attributes and architecture of software intensive systems. In *PROGRESS Symposium 2003*, PROGRESS, October 2003. 22

E. R. Poort and H. van Vliet. Architecting as a risk- and cost management discipline. In *Proceedings 9th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pages 2–11. IEEE Computer Society, 2011. 129

E. R. Poort, H. Postema, A. Key, and P. H. de With. The influence of CMMI on establishing an architecting process. In *Third International Conference on the Quality of Software-Architectures (QoSA)*, 2007. 79, 91

J. D. Procaccino. What do software practitioners really think about project success: an exploratory study. *Journal of Systems and Software*, 78:194–203, 2005. 61

Z. Racheva, M. Daneva, and A. Herrmann. A conceptual model of client-driven agile requirements prioritization: results of a case study. In *2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '10, pages 39:1–39:4, New York, NY, USA, 2010. ACM. 142

RACV Insurance Pty Ltd v. Unisys Australia Ltd. RACV Insurance Pty Ltd v. Unisys Australia Ltd, 2001. VSC 300. 41

B. Regnell, R. Berntsson Svensson, and T. Olsson. Supporting roadmapping of quality requirements. *IEEE Software*, 25:42–47, March 2008. 43, 52, 141

S. Robertson and J. Robertson. *Mastering the Requirements Process (2nd Edition)*. Addison-Wesley Professional, 2006. 35

SHARK. 4th workshop on sharing and reusing architectural knowledge. IEEE Computer Society, 2009. 117, 122

M. Shaw. Toward higher-level abstractions for software systems. *Data & Knowledge Engineering*, 5(2):119 – 128, 1990. 3, 121, 123

H. Simon. *The Sciences of the Artificial*. MIT Press, 1969. 140

O. P. N. Slyngstad, R. Conradi, M. A. Babar, V. Clerc, and H. van Vliet. Risks and risk management in software architecture evolution: An industrial survey. In *15th Asia-Pacific Software Engineering Conference (APSEC)*, pages 101–108, 2008. 139, 143

Standish Group. *Chaos Report*. 1994. 118

S. Supakkul, T. Hill, L. Chung, T. T. Tun, and J. C. S. do Prado Leite. An NFR pattern approach to dealing with NFRs. In *2010 18th IEEE International Requirements Engineering Conference*, RE '10, pages 179–188, Washington, DC, USA, 2010. IEEE Computer Society. 52

A. Sutcliffe. The socio-economics of software architecture. *Automated Software Engineering*, 15:343–363, 2008. 176

A. Tang and J. Han. Architecture rationalization: A methodology for architecture verifiability, traceability and completeness. In *12th Annual IEEE International Conference on the Engineering of Computer-Based Systems (ECBS)*, pages 135–144, 2005. 140

The Open Group. The open group certified architect (Open CA) program – conformance requirements (multi-level). 176

The Open Group. The open group architecture framework (TOGAF), 2009. URL `http://www.togaf.info`. 4

J. Tyree and A. Akerman. Architecture decisions: Demystifying architecture. *IEEE Software*, 22(2):19–27, 2005. 3, 81, 90, 101, 121, 122, 123, 149, 151

US Department of Commerce. Enterprise architecture capability maturity model. online, 2007. URL `http://ocio.os.doc.gov/ITPolicyandPrograms/Enterprise_Architecture/PROD01_004934`. retrieved 8 Jan 2012. 99, 176

US Government. Code of federal regulations, title 48: Federal acquisition regulation. National Archives and Records Administration, 2005. 38, 49

J. S. van der Ven, A. Jansen, P. Avgeriou, and D. K. Hammer. Using architectural decisions. In *Second International Conference on the Quality of Software Architectures (QoSA 2006)*, LNCS, 2006. 81

A. van Lamsweerde. *Requirements Engineering – From System Goals to UML Models to Software Specifications*. Wiley, 2009. 35

J. C. Westland. The cost of errors in software development: evidence from industry. *Journal of Systems and Software*, 62(1):1–9, May 2002. 62

K. E. Wiegers. *Software Requirements, Second Edition (Pro-Best Practices)*. Microsoft Press, 2 sub edition, 2003. 35

O. Zimmermann, T. Gschwind, J. Küster, F. Leymann, and N. Schuster. Reusable architectural decision models for enterprise application development. In *Third International Conference on the Quality of Software Architectures (QoSA)*, number 4880/2008 in LNCS, pages 157–166. Springer, 2007. 122, 150