RCDA: Architecting as a Risk- and Cost Management Discipline

Eltjo R. Poort[a,*], Hans van Vliet[b]

[a] *Logica, Amstelveen, The Netherlands*
[b] *VU University Amsterdam, De Boelelaan 1081, 1081 HV Amsterdam, The Netherlands*

Abstract

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. The point of view is the basis of RCDA, the Risk- and Cost Driven Architecture approach. So far, more than 150 architects have received RCDA training. For a majority of the trainees, RCDA has a significant positive impact on their architecting work.

Keywords: Software architecture, Risk Management, Cost management

1. Introduction

The notion of "software architecture" is one of the key technical advances in the field of software engineering over the last decades [1]. 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 [2, 3]
2. Architecture as a set of design decisions, including their rationale [4, 5, 6]

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" [2]. View 1 is focused on the choice and organization of components and connectors, but practicing architects' decisions appear to have a much wider range.

*Corresponding author
Email addresses: eltjo.poort@logica.com (Eltjo R. Poort), hans@cs.vu.nl (Hans van Vliet)

View 2, architecture as a set of design decisions [5], is more generic and has been beneficial to both the architecture research community and its practitioners [6]. 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 [6, 7], and it has stimulated research into re-usable architectural decisions [8, 9]. 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 [10], 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 paper, 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

This point of view is the basis of an architecting approach developed in Logica: Risk- and Cost Driven Architecture (RCDA), discussed in section 7. RCDA encompasses a lot more than merely viewing architecture as a risk- and cost-driven activity. It consists of a set of practices covering the whole spectrum of activities involved in architecting. So far, more than 150 architects have received RCDA training. We have assessed the impact hereof on the work of the architects. It shows that for a majority of the trainees, RCDA has a significant positive impact on their architecting work.

The rest of this paper is structured as follows:

- in section 2, we will show how we arrived at this point of view of what architecture is all about, and define some key concepts
- in section 3, we will elaborate on the meaning of risk and cost and how they determine architectural significance
- then in section 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 section 5, we will use three examples from industry to illustrate how this approach helps to communicate architectural significance to stakeholders
- in section 6, we will give some guidance on how to implement this point of view
- in section 7, we give an overview of RCDA, the Risk- and Cost Driven Architecture Approach
- in section 8, we give quantitative and qualitative evidence of the impact of RCDA
- we conclude the paper with a discussion, including some frequently asked questions about this approach

This paper is an extended version of [11]. The main extensions consist of the description of RCDA in section 7 as well as the empirical validation of its impact in section 8. We have also added more depth to the discussion on risk and the related work section.

2. What are architectural decisions about?

For years, we have been talking and writing about *software* architecture as a set of *design* decisions [4, 5, 6]. 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 [12], 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 [2]. 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 [13].

Let's see if the international standard definition of software architecture [12] 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 [13], 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. This point of view is the basis for RCDA, the Risk- and Cost Driven Architecture Approach. 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 sections 4.2 and 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 section 5.

2.1. Key concepts

"Concern" and "Decision" are key concepts throughout this paper. Concern is a well-understood, established concept: [12] 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 [14] 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 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

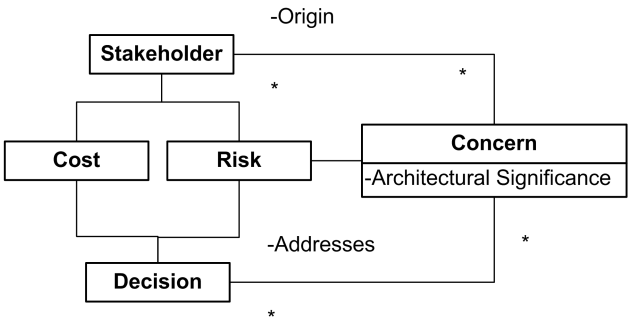


Figure 1: Key concept model.

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 [15, 5, 12]. [15, 5] 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 [12] we simplify and generalize them to Figure 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 paper to describe the amount of attention the architect should pay to a particular concern.

3. Architectural significance in terms of risk and cost

In section 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 section 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 section 3.2. $Risk(C)$ is the total expected cost of "things going wrong" associated with C as explained in section 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 section 3.1.1.

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 [16]. 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 [17] risk-driven model; we add cost as an equally prominent factor to consider.

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 paper is the risk exposure relationship used by Boehm [18]:

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

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 [19], 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.

3.1.1. 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.

3.1.2. 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 [11] 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." [13]. It also resonates strongly with [20], who state that "the software architecture of deployed software is determined by those aspects that are the hardest to change."

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. [21]), 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.

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 use the generic approach documented in [22]. In this paper, Hofmeister et al. compare five industrial approaches to architectural design, and extract 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.

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 [22], the ASR definition is borrowed from [23]: "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.

4.2. Architecting workflow

In [22], 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 section 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."[22]. 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.

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.

The time at which to address architectural concerns is influenced by their risk-related character. We have seen in section 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 Figure 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

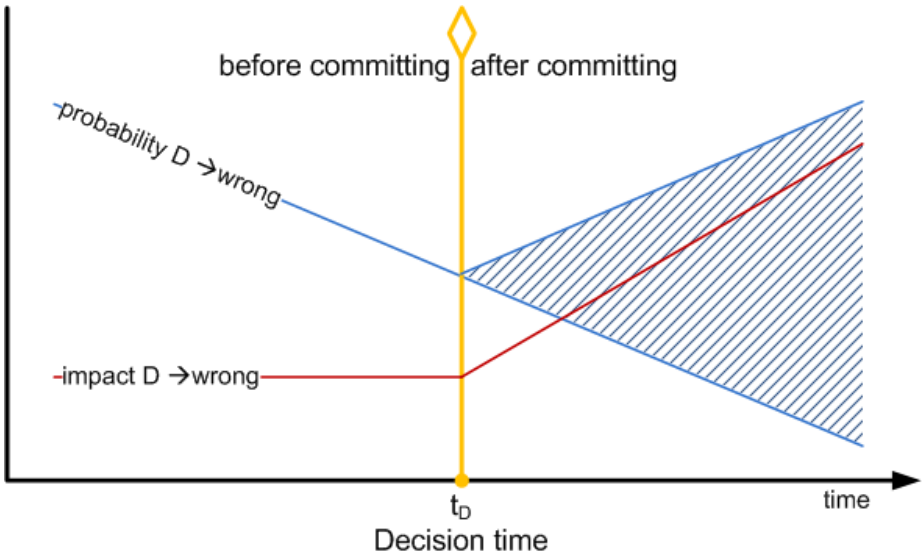


Figure 2: Decision risk factors over time.

- *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 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 [24] using decision trees and real-option theory.

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 [25]. 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 give the architect a relatively objective measure to explain priorities to stakeholders. In section 5.1, we will list some 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 [26] architects to projects or product organizations.

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

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

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.

- **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.

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 [27]. Architects invariably have a limited amount of time and they should spend it addressing concerns with the most pressing risks [17] and cost.

7. The RCDA Approach

This section describes RCDA, the Risk and Cost Driven Architecture approach developed in Logica. The approach consists of a set of practices, harvested from Logica practitioners and enhanced by our research presented in earlier sections of this paper. We present the structure of the approach and its rationale.

Of the architects we have taught the approach so far, 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 process involved all the same key concepts: stakeholders, concerns, decision making, etc. All of the non-software- architects indicated that the principles presented, based on viewing architecture as a risk- and cost management discipline, 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) [33] 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.

7.1. RCDA Practices

The basic building blocks of RCDA are *practices*. A practice is a way to systematically characterize a problem and address it [28]. 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 and prioritizing architecturally significant requirements and concerns, according to the principles laid out in section 3. Of all RCDA practices, this practice most strongly represents the risk- and cost driven view.

Dealing with Non-Functional Requirements gives guidance on identifying and addressing the risks associated with NFRs, which are often underexposed and can have major impact on the solution. Risk assessment is key in this practice.

Stakeholder Workshop is a practice for obtaining architectural requirements from stakeholders, based on the SEI's Quality Attribute Workshop [29].

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, taking risk and cost into account.

Solution Shaping Workshop is a special case of a Stakeholder Workshop for a fixed price bid or project. All delivery 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 [10].

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. [14, 30].

Architecture Documentation documents the current state of the solutions architecture in a set of views [31, 12], 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. [6, 5]. The risk/cost perspective adds tracking of risks and cost associated with the concerns and decisions.

Solution Costing gives guidance on early costing of delivering a solution using a selected architecture. This core practice is introduced through the risk/cost perspective.

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 [32], enhanced by the risk/cost perspective.

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

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

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, identifying and handling the associated risk and cost.

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 risk, cost and benefit of architectural requirements.

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 [9] and [33].

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 [34].

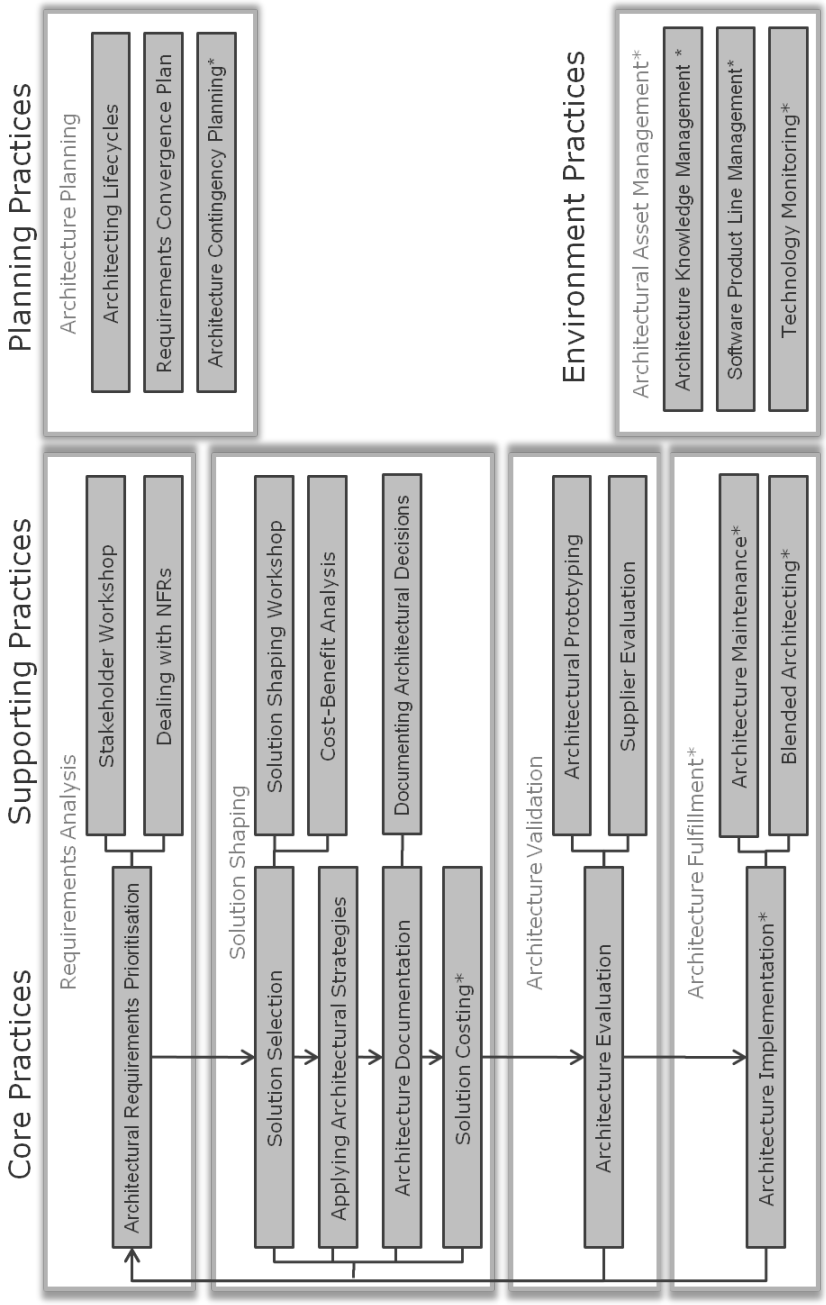


Figure 3: RCDA Practices organized by practice set and practice category.

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 Figure 3. In a second dimension, Figure 3 organizes the practices into the following four categories:

Core practices We chained 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.

Figure 3 shows 22 practices: 14 that are part of the current version of RCDA, 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.

7.2. 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, 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 the preceeding sections. 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 [27] and [17]. 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." [35] It is applied through the Architectural Requirements Prioritization practice.

Architecture as both blueprint and design decisions is based on papers such as [5, 6]. 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 [13] and [25]. 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.

8. 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.

8.1. 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 1 lists the practices in RCDA. 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 1 both before and after receiving the training:

- 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)

8.2. 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.

Table 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

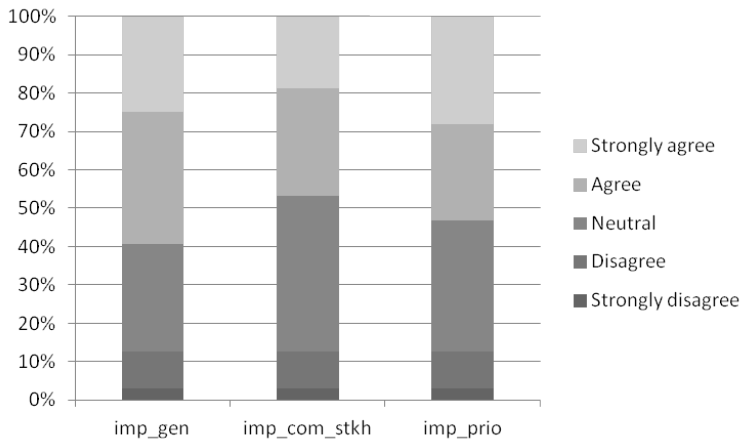


Figure 4: Three statements on effectiveness of RCDA

- 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.

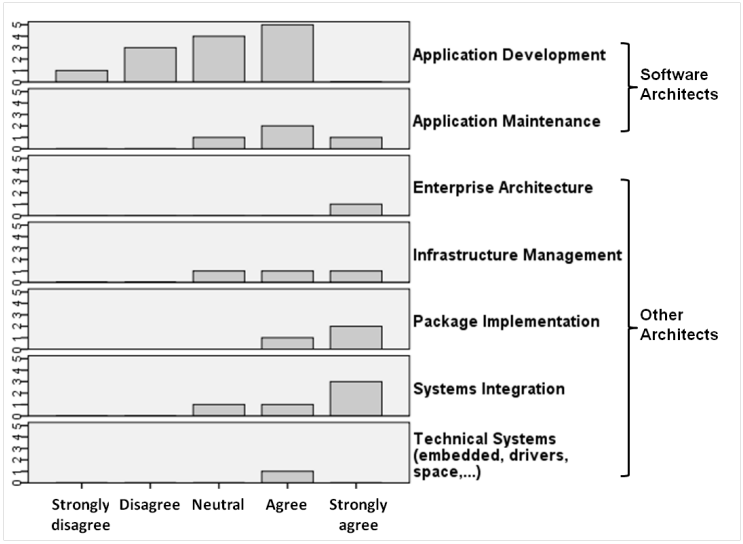
Figure 4 shows the responses to the three general statements about the effectiveness of RCDA (Section C):

imp_gen *In general, my effectiveness as an architect has improved after being trained in RCDA.*

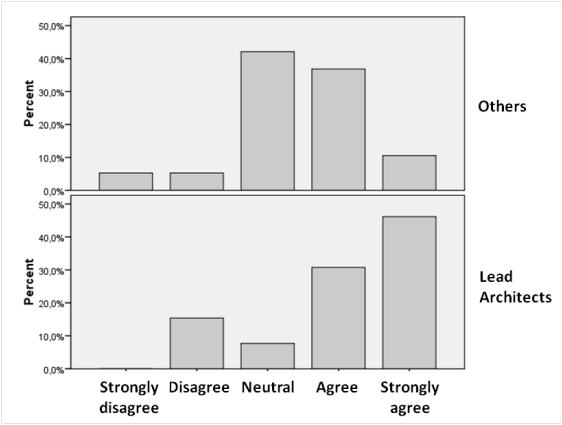
imp_com_stkh *RCDA helps me to communicate with stakeholders more effectively.*

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 section 1. Less then 15% disagree with any of the statements, but a significant portion of the trainees answered "neutral", implying they could not decide whether or not the training had made them more effective. Adding the "disagree" to the "neutral" responses, we get 40% of respondents not agreeing that the training had made them more effective. Figures 5a and 5b shed some light on this significant number by breaking down the respondents.



(a) By architecture genre



(b) Lead architects vs others

Figure 5: Agreement with "In general, my effectiveness as an architect has improved after being trained in RCDA."

In Figure 5a, 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 *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.

Figure 5b 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).

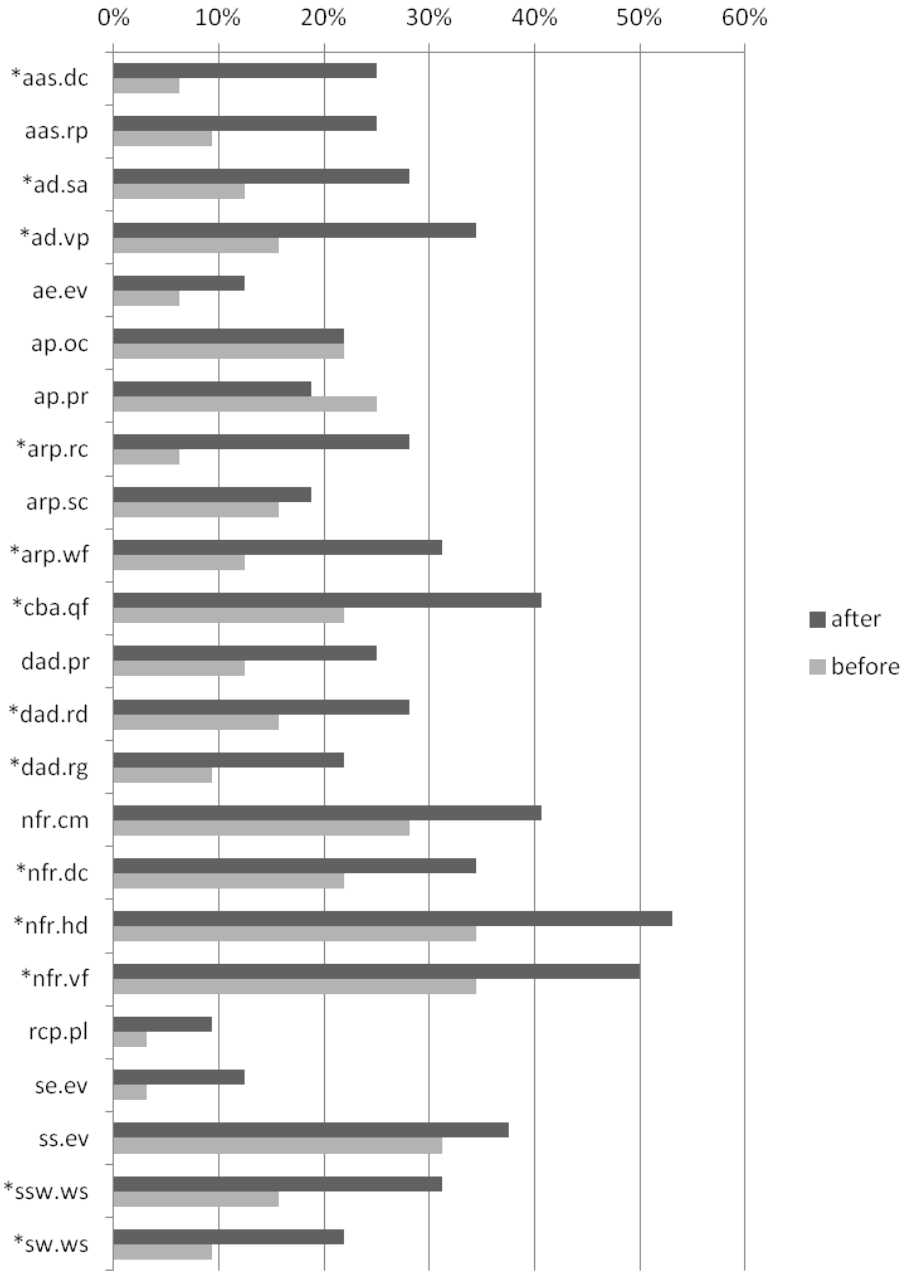
8.2.1. RCDA Principles

We asked the architects about the frequency with which they applied the general RCDA principles explained in section 7.2, and the impact. Table 2 shows the results. Column "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.

8.2.2. RCDA Practices

Figure 6 shows the percentage of respondents indicating they have applied the key guidance elements of the RCDA practices listed in Table 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



* Increase in application frequency significant at 0.05 level

Figure 6: RCDA practices: respondents applying guidance before and after training (abbreviations on p.24)

Table 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

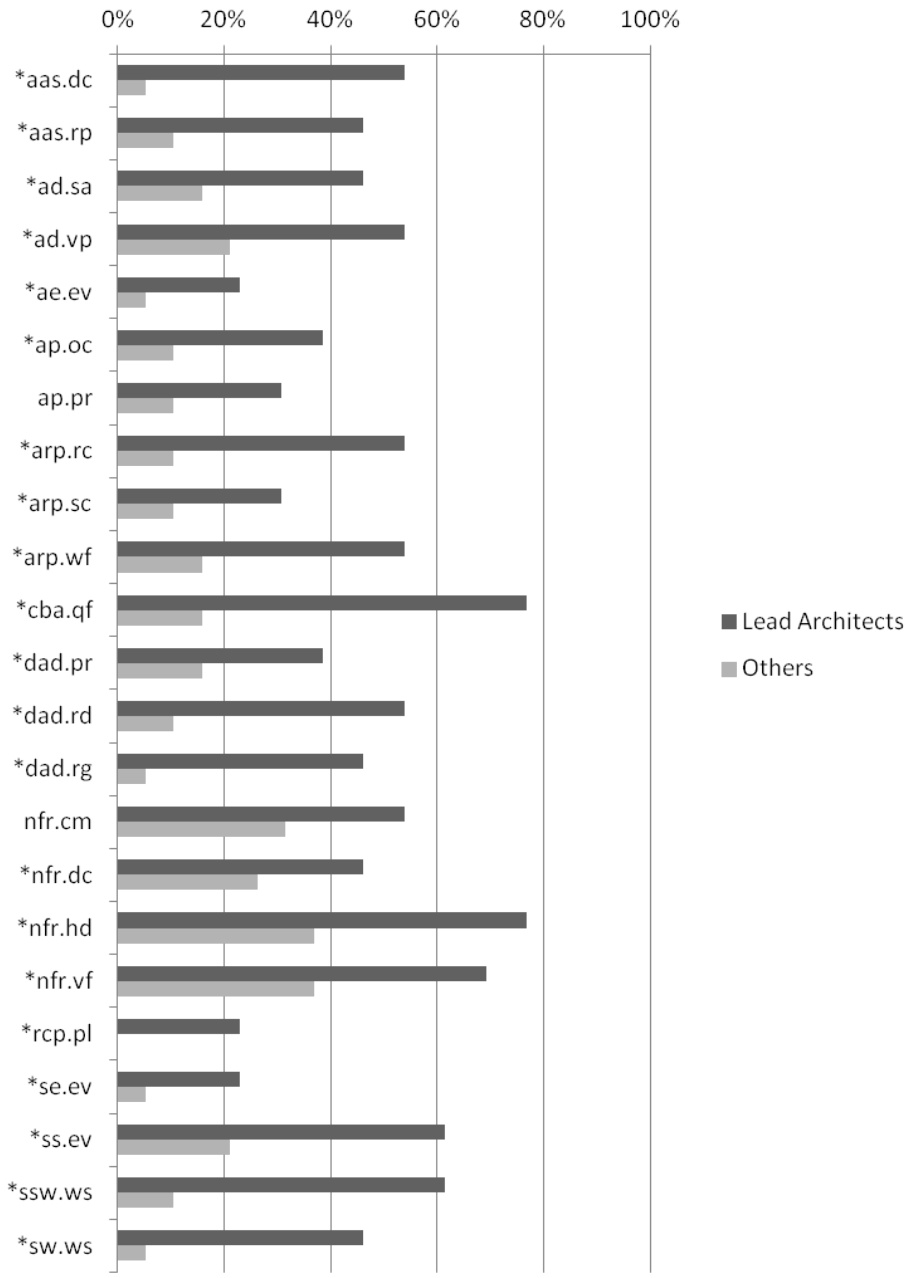
compare the percentages for lead architects versus other respondents, as visualized in Figure 7. We see that more lead architects than others are applying the guidance, and almost half of the guidance elements are applied by the majority of the lead architects. Analysis shows that the "lead architect" responses are significantly correlated to the "frequency applied after training" responses for all guidance elements except ap.pr and nfr.cm (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.

Figure 8 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 Figure 6). 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



* Correlation with lead architect response significant at 0.05 level

Figure 7: RCDA practices: lead architects vs others applying guidance after training (abbreviations on p.24)

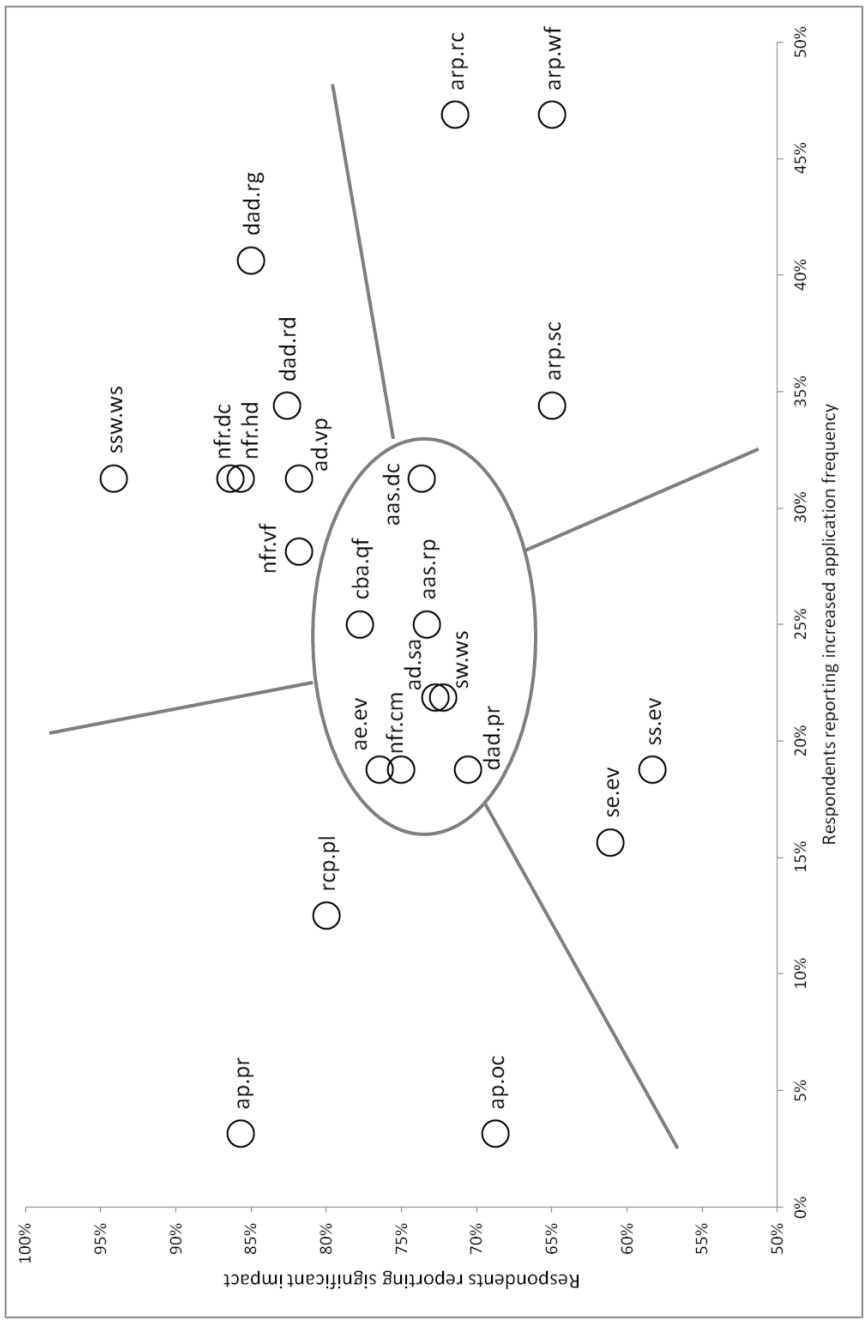


Figure 8: RCDA practices: impact vs. increased application after training (abbreviations on p.24)

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 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 Figure 8). 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).

8.3. Discussion

The sparse application and relatively low appreciation of formal evaluation practices like se.ev and ss.ev (Figure 8) is in line with findings by e.g. [36], which reports that "methods and techniques to validate the architecture ... are not embedded within the mindset of architects." Another finding from the [36] 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.

8.3.1. 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 Figure 8, ap.pr is in the top left cluster with requirements convergence planning, another practice that requires significant planning and use of resources.

8.3.2. 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 section 7.2), so that those who apply RCDA tend to take more ownership and responsibility, which puts them in leadership positions.

Another effect may be that the lead architects have to deal more with the consequences of cost and risk, and hence are more sensitive to these factors. The data set did not provide any means to confirm or reject any of the explanations: they may well all be valid, and reinforce each other's impact.

The relatively small impact on perceived effectiveness of non-lead architects suggests that the efficiency of the course may be increased by tighter selection of students, particularly their expected leading roles. Non-lead architects may be helped more by guidance on *understanding* architectural decisions than by guidance on prioritizing and making these decisions.

8.4. 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.

9. Related work

9.1. 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 [37]. The importance of risk analysis in software development is aptly phrased by Tom Gilb [38]: "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 [18]. 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?" [39]. Another class of papers discusses sophisticated techniques for computationally handling risks, using Bayesian networks, fuzzy set theory, and the like; [40] 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 [41], 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, [42] 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) [14] is a list of risks and non-risks. By studying the output of a series of such ATAM evaluations, [43] were able to reveal and analyze risk themes specifically geared towards software architecture. [44] 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.

9.2. 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) [10] and the Architecture Rationalization Method (ARM) [45] that relate architectural decisions to the benefits they bring to an organization, studies that emphasize business implications of architectural decisions [25], and approaches that consider architectural decisions as investment decisions [24, 46].

Feather et al. [47] 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 [17] 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 [48] 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 [49, 50]. 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.

9.3. 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 [51]: "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 [52], but often other factors are also taken into account, such as return on investment (RoI) [10, 53] and risk (mainly in security-oriented requirements engineering [54]). Our failure scenarios in section 3.1 are strongly related to the Misuse Cases found in MOQARE [55]. Based on priority, RPMs allocate requirements to deliver iterations in software projects, or releases in product roadmaps. Herrmann et al. [56] examine 15 RPMs, and analyzes their use of benefit and cost as prioritizing factors. Racheva et al. [57] derive 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 paper: 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 [58], "certain classes of systems that ignore architectural issues for too long hit a wall and collapse due to a lack of an architectural focus."

10. Frequently Asked Questions

In this section we discuss a number of questions that were frequently raised when teaching the approach to practicing architects. Since they led to interesting discussions, we are presenting them here.

10.1. 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 architecturally 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. [20], 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 3.1. Slyngstad et al. [44] have surveyed risks in software architecture evolution, and present the most common risks for architectures of existing systems and their associated mitigation activities.

10.2. 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 predetermining 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.

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 [10], a method for performing economic modeling of software systems, centered on an analysis of their architecture.
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 paper, however, is on *how* to build a solution, and then risk and cost trump value.

As explained above, Kahneman [50] 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.

10.3. 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.

11. 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.

The risk- and cost driven view on architecture is the basis for RCDA, the Risk- and Cost Driven Architecture Approach. It is part of a solution architecture training programme that has so far been taught to 159 architects. The results of a survey amongst these architects 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, but its effectiveness may be improved by selecting students that are expected to fulfill lead architect roles shortly after the training.

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

References

[1] R. Farenhorst, R. de Boer, Architectural Knowledge Management - Supporting Architects and Auditors, Ph.D. thesis, VU University Amsterdam (2009).

[2] M. Shaw, Toward higher-level abstractions for software systems, Data & Knowledge Engineering 5 (2) (1990) 119 – 128.

[3] D. E. Perry, A. L. Wolf, Foundations for the study of software architecture, SIGSOFT Softw. Eng. Notes 17 (1992) 40–52.

[4] P. Kruchten, The Rational Unified Process: An Introduction, Addison-Wesley, Boston, 1998.

[5] A. Jansen, J. Bosch, Software Architecture as a Set of Architectural Design Decisions, in: 5th Working IEEE/IFIP Conference on Software Architecture, IEEE Computer Society, Washington, DC, USA, 2005, pp. 109–120.

[6] J. Tyree, A. Akerman, Architecture decisions: Demystifying architecture, IEEE Software 22 (2) (2005) 19–27.

[7] A. Ivanović, P. America, Information needed for architecture decision making, in: 2010 ICSE Workshop on Product Line Approaches in Software Engineering, PLEASE '10, ACM, New York, NY, USA, 2010, pp. 54–57.

[8] SHARK, 4th Workshop on SHAring and Reusing architectural Knowledge, IEEE Computer Society (2009).

[9] O. Zimmermann, T. Gschwind, J. Küster, F. Leymann, N. Schuster, Reusable Architectural Decision Models for Enterprise Application Development, in: Third International Conference on the Quality of Software Architectures (QoSA), no. 4880/2008 in LNCS, Springer, 2007, pp. 157–166.

[10] R. Kazman, J. Asundi, M. Klein, Making Architecture Design Decisions: An Economic Approach, Tech. Rep. CMU/SEI-2002-TR-035, SEI (2002).

[11] E. R. Poort, hans van Vliet, Architecting as a Risk- and Cost Management Discipline, in: Proceedings 9th Working IEEE/IFIP Conference on Software Architecture (WICSA), IEEE Computer Society, 2011, pp. 2–11.

[12] ISO 42010, Systems and software engineering Architecture description, iSO 42010:2011 (2011).

[13] M. Fowler, Who needs an Architect?, IEEE Software 20 (5) (2003) 11–13.

[14] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, 2nd. ed., Addison Wesley, 2003.

[15] R. de Boer, R. Farenhorst, P. Lago, H. van Vliet, V. Clerc, A. Jansen, Architectural Knowledge: Getting to the Core, in: S. Overhage, C. Szyperski, R. Reussner, J. Stafford (Eds.), Software Architectures, Components, and Applications, Vol. 4880 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2007, pp. 197–214.

[16] M. Glinz, A Risk-Based, Value-Oriented Approach to Quality Requirements, IEEE Software 25 (2008) 34–41.

[17] G. Fairbanks, Just Enough Architecture: The Risk-Driven Model, Crosstalk.

[18] B. W. Boehm, Software Risk Management: Principles and Practices, IEEE Software 8 (1991) 32–41.

[19] AACE, Risk Analysis and Contingency Determination using Range Estimating, AACE International Recommended Practice No. 41R-08 (2000).

[20] A. Klusener, R. Lämmel, C. Verhoef, Architectural modifications to deployed software, Science of Computer Programming 54 (2005) 143–211.

[21] B. Boehm, Software Engineering Economics, Prentice Hall, 1981.

[22] C. Hofmeister, P. Kruchten, R. L. Nord, J. H. Obbink, A. Ran, P. America, A general model of software architecture design derived from five industrial approaches, Journal of Systems and Software 80 (1) (2007) 106–126.

[23] H. Obbink, P. Kruchten, W. Kozaczynski, R. Hilliard, A. Ran, H. Postema, L. Dominick, R. Kazman, W. Tracz, E. Kahane, Report on Software Architecture Review and Assessment (SARA), Tech. rep., SARA Working group, retrieved 11 January 2012 (2002).
URL `http://kruchten.com/philippe/architecture/SARAv1.pdf`

[24] S. Biffl, A. Aybuke, B. Boehm, H. Erdogmus, P. Gruenbacher (Eds.), Value-Based Software Engineering, Springer, 2006, Ch. Valuation of Software Initiatives Under Uncertainty: Concepts, Issues, and Techniques, pp. 39–66.

[25] P. Clements, R. Kazman, M. Klein, D. Devesh, S. Reddy, P. Verma, The Duties, Skills, and Knowledge of Software Architects, in: 6th Working IFIP/IEEE Conference on Software Architecture (WICSA), IEEE Computer Society, 2007.

[26] Money Magazine, Best Jobs in America 2010, Top 100, CNN on-line, rank 1: Software Architect (November 2010).
URL `http://money.cnn.com/magazines/moneymag/bestjobs/2010/snapshots/1.html`

[27] R. Malan, D. Bredemeyer, Less is more with minimalist architecture, IT Pro (2002) 46–48.

[28] I. Jacobson, P. W. Ng, I. Spence, Enough process - let's do practices, Journal of Object Technology 6 (6) (2007) 41–66.

[29] M. R. Barbacci, R. Ellison, A. J. Lattanze, J. A. Stafford, C. B. Weinstock, W. G. Wood, Quality Attribute Workshops (QAWs), Third Edition, Tech. Rep. CMY/SEI-2003-TR-016, SEI (2003).

[30] E. Gamma, R. Helm, R. E. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, Reading, MA, 1995.

[31] P. Kruchten, The 4+1 view model of architecture, IEEE Software 12 (6) (1995) 45–50.

[32] G. Abowd, L. Bass, P. Clements, R. Kazman, L. Northrop, A. Zaremski, Recommended Best Industrial Practice for Software Architecture Evaluation, Tech. Rep. CMU/SEI-96-TR-025, SEI (1997).

[33] R. Farenhorst, H. van Vliet, Experiences with a wiki to support architectural knowledge sharing.

[34] P. Clements, L. Northrop, Software Product Lines, Addison Wesley, 2002.

[35] R. Kazman, L. Bass, M. Klein, The essential components of software architecture design and analysis, Journal of Systems and Software 79 (8) (2006) 1207–1216.

[36] V. Clerc, P. Lago, H. van Vliet, The architect's mindset, in: Quality of software architectures 3rd international conference on Software architectures, components, and applications, QoSA'07, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 231–249.

[37] P. Bannerman, Risk and risk management in software projects: A reassessment, Journal of Systems and Software 81 (2008) 2118–2133.

[38] T. Gilb, Principles of Software Engineering Management, Addison Wesley, 1988.

[39] H. R. Costa, M. de O. Barros, G. H. Travassos, Evaluating software project portfolio risks, Journal of Systems and Software 80 (1) (2007) 16–31.

[40] H. Lee, Applying fuzzy set theory to evaluate the rate of aggregate risk in software development, Fuzzy Sets and Systems 80 (3) (1996) 261–271.

[41] B. Boehm, A Spiral Model of Software Development and Enhancement, IEEE Computer 21 (5) (1988) 61–72.

[42] B. Boehm, R. Turner, Balancing Agility and Discipline, Addison Wesley, 2004.

[43] L. Bass, R. Nord, W. Wood, D. Zubrow, Risk Themes Discovered Through Architecture Evaluations, in: 6th Working IFIP/IEEE Conference on Software Architecture (WICSA), IEEE Computer Society, 2007, pp. 1–10.

[44] O. P. N. Slyngstad, R. Conradi, M. A. Babar, V. Clerc, H. van Vliet, Risks and Risk Management in Software Architecture Evolution: An Industrial Survey, in: 15th Asia-Pacific Software Engineering Conference (APSEC), 2008, pp. 101–108.

[45] A. Tang, 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), 2005, pp. 135–144.

[46] A. Ivanović, P. America, Strategy-focused Architecture Decision Making, in: P. van de Laar, T. Punter (Eds.), Views on Evolvability of Embedded Systems, Springer, 2010, pp. 245–260.

[47] M. S. Feather, S. L. Cornford, K. A. Hicks, J. D. Kiper, T. Menzies, A broad, quantitative model for making early requirements decisions, IEEE Software 25 (2008) 49–56.

[48] H. Simon, The Sciences of the Artificial, MIT Press, 1969.

[49] D. Kahneman, A. Tversky, Prospect Theory: An analysis of Decision and Risk, Econometrica 47 (1979) 263–291.

[50] D. Kahneman, Thinking, Fast and Slow, Farrar, Straus and Giroux, 2011.

[51] Agile Alliance, Manifesto for Agile Software Development, http://agilemanifesto.org (2001).

[52] T. Gilb, Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage, Butterworth-Heinemann, Newton, MA, USA, 2005.

[53] B. Regnell, R. Berntsson Svensson, T. Olsson, Supporting roadmapping of quality requirements, IEEE Software 25 (2008) 42–47.

[54] A. Herrmann, A. Morali, 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).

[55] A. Herrmann, B. Paech, Moqare: misuse-oriented quality requirements engineering, Requir. Eng. 13 (2008) 73–86. doi:10.1007/s00766-007-0058-9.

[56] A. Herrmann, M. Daneva, Requirements prioritization based on benefit and cost prediction: An agenda for future research, in: 2008 16th IEEE International Requirements Engineering Conference, IEEE Computer Society, Washington, DC, USA, 2008, pp. 125–134.

[57] Z. Racheva, M. Daneva, 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, ACM, New York, NY, USA, 2010, pp. 39:1–39:4.

[58] P. Abrahamsson, M. A. Babar, P. Kruchten, Agility and architecture: Can they coexist?, IEEE Software 27 (2010) 16–22.