8 Architecting as a Risk- and Cost Management Discipline

We propose to view architecting as a risk- and cost management discipline. This point of view helps architects identify the key concerns to address in their decision making, by providing a simple, relatively objective way to assess architectural significance. It also helps business stakeholders to align the architect's activities and results with their own goals. We examine the consequences of this point of view on the architecture process, and give some guidance on its implementation, using examples from practicing architects trained in this approach.

8.1 Introduction

As mentioned in the introduction to this thesis, the notion of "software architecture" is one of the key technical advances in the field of software engineering over the last decades [Farenhorst and de Boer, 2009]. In that period, there have been two distinct fundamental views as to the nature of architecture:

1. Architecture as a higher level abstraction for software systems, expressed in components and connectors [Shaw, 1990, Perry and Wolf, 1992].
2. Architecture as a set of design decisions, including their rationale [Kruchten, 1998, Jansen and Bosch, 2005, Tyree and Akerman, 2005].

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

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

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

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

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

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

The rest of this chapter is structured as follows:

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

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

8.2 What are Architectural Decisions About?

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

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

Let's see if the international standard definition of software architecture [ISO 42010, 2011] helps: "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution". Going back to the example of our programmer and his daily minor design decisions, these can certainly be about the properties of the system, its elements and their relationships. The programmer's decisions may also affect the system's evolution. There are only two concepts in the ISO 42010 definition that are clearly out of the league of the programmer's decisions: "fundamental" and "principles". These concepts can guide us towards the class of decisions architects should focus on: decisions about fundamental things and principles. Ralph Johnson, quoted by Martin Fowler [Fowler, 2003], makes a similar statement in less formal words: "Architecture is about the important stuff. Whatever that is."

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

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

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

8.2.1 Key concepts

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

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

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

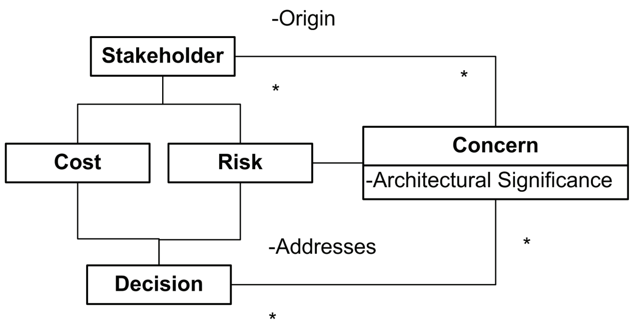

Figure 8.1: Key concept model.

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

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

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

The final concept we would like to define here is Architectural Significance. This is an attribute of Concern. It is the key concept used in this chapter to describe the amount of attention the architect should pay to a particular concern.

8.3 Architectural Significance in Terms of Risk and Cost

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

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

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

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

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

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

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

8.3.1 Risk

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

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

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

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

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

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

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

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

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

Risk example

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

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

F_1 The hosting platform is underdimensioned.

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

F_3 The number of web-users exceeds expectations.

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

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

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

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

Decision risk

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

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

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

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

8.3.2 Cost

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

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

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

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

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

8.4 Impact on Architecting Process

We will now examine the impact of the risk- and cost management view of architecture on the architecting process. In the previous sections, we have discussed the importance of managing risk and cost in architecture, and presented a method for ordering concerns by architectural significance. In this section, we will show how this ordering can be used to optimize an architecting process so that it becomes better at controlling risk and cost.

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

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

8.4.1 Architecting activities

The generalized architecting activities are:

1. *Architectural analysis*: define the problems the architecture must solve. This activity examines architectural concerns and context in order to come up with a set of Architecturally Significant Requirements (ASRs).
2. *Architectural synthesis*: the core of architecture design. This activity proposes architecture solutions to a set of ASRs, thus it moves from the problem to the solution space.
3. *Architectural evaluation*: ensures that the architectural design decisions made are adequate. The candidate architectural solutions are measured against the ASRs.

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

8.4.2 Architecting workflow

In [Hofmeister et al., 2007], the authors describe an "apparently haphazard process" in which architects maintain, implicitly or explicitly, a "backlog of smaller needs, issues, problems they need to tackle and ideas they may want to use. The backlog drives the workflow, helping the architect determine what to do next." Hofmeister et al. make clear that the backlog typically consists of architectural concerns but also other types of items, and that it is constantly re-prioritized using various prioritization tactics. They mention risk as one of the mostly external forces driving priority; others are team or stakeholder pressure or perception of greater difficulty.

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

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

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

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

8.4.3 Architectural decisions and the flow of time

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

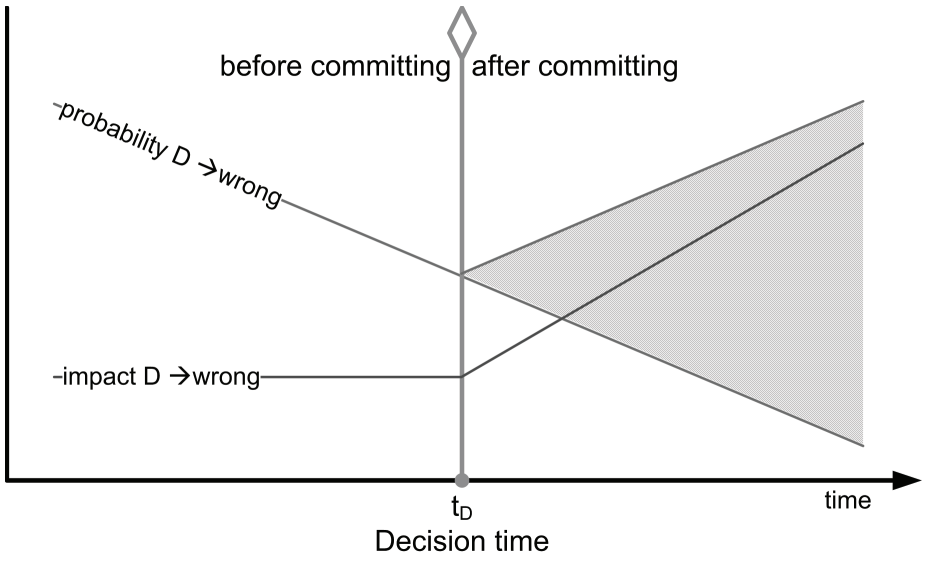

Figure 8.2: Decision risk factors over time.

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

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

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

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

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

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

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

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

8.5 Stakeholder Communication

Many stakeholders, especially business managers, are not used to thinking and talking in terms of levels of abstraction or components and connectors. Part of the architect's job is to translate architectural concerns and decisions into terms that stakeholders understand [Clements et al., 2007]. One substantial advantage of expressing architectural significance in terms of risk and cost is that they are universal terms that most stakeholders can relate to. These terms smooth communication between architect and stakeholders, and gives the architect a relatively objective measure to explain priorities to stakeholders. We already saw the importance of objectivity in stakeholder communication about solution architecture in the SMSC case study in Chapter 5. In §8.5.1, we will list some more examples from practice of architectural concerns and how they can be expressed in terms of risk and cost to facilitate stakeholder communication.

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

8.5.1 Examples from practicing architects

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

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

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

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

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

8.6 Implementing the Risk- and Cost Driven View of Architecting

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

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

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

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

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

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

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

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

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

8.7 Conclusions and Discussion

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

8.7.1 Related work

Risk in software architecture

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

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

Risk and cost in decision making

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

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

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

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

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

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

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

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

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

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

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

Requirements prioritization

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

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

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

A well-known requirements prioritization principle comes from the Agile Manifesto [Agile Alliance, 2001]: "Our highest priority is to satisfy the customer through early and continuous delivery of valuable software". Thus, many RPMs use business value as the main prioritizing factor [Gilb, 2005], but often other factors are also taken into account, such as return on investment (RoI) [Kazman et al., 2002, Regnell et al., 2008] and risk (mainly in security-oriented requirements engineering [Herrmann et al., 2010]). Our failure scenarios in §8.3.1 are strongly related to the Misuse Cases found in MOQARE [Herrmann and Paech, 2008]. Based on priority, RPMs allocate requirements to deliver iterations in software projects, or releases in product roadmaps. [Herrmann and Daneva, 2008] examines 15 RPMs, and analyzes their use of benefit and cost as prioritizing factors. [Racheva et al., 2010] derives a conceptual model for requirements prioritization in an agile context based on 19 RPMs. The model identifies five requirements prioritizing aspects for stakeholders: Business Value, Risk, Effort Estimation/ Size Measurement, Learning Experience, and External Change. We recognize the two main themes of this chapter: Risk and Cost. For a discussion on Business Value in architecting, please refer to the next section. For architects, the other two factors are an integral part of their architecting process: External Change is usually handled as a modifiability concern and a risk factor, while Learning Experience is the means by which architects address uncertainty in the solution domain, using practices like architectural prototyping.

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

8.7.2 Frequently Asked Questions

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

What about existing systems?

What does our view on architecture mean for existing systems, i.e. systems after their initial delivery? Since we already know how the architectural decisions made during the design phase turned out, does it still make sense to talk about the risk of making wrong decisions? It does: just like during the design phase, the architectural activities related to existing systems have risk and cost management as their prime business objective. Typical activities at this stage are architecture recovery, evaluation and architectural modifications to the system. The reason an architect gets involved is to identify architecturally significant concerns related to something that the stakeholders want to do with this system; most likely, modify it in some way. And once again, what is 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., 2005], in their extensive paper on modifying existing systems, come to a very similar notion of architectural significance in those systems, as we have seen in §8.3.1. [Slyngstad et al., 2008] have surveyed risks in software architecture evolution, and present the most common risks for architectures of existing systems and their associated mitigation activities.

What about value?

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

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

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

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

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

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

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

Does this mean architects always have to minimize risks?

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

8.7.3 Conclusion

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

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

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