Architecting as a Risk- and Cost Management Discipline

Eltjo R. Poort
Logica
Amstelveen, The Netherlands
eltjo.poort@logica.com

Hans van Vliet
VU University
Amsterdam, The Netherlands
hans@cs.vu.nl

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, and give some guidance on its implementation, using examples from practicing architects trained in this approach.

I. 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]

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.

View 2, architecture as a set of design decisions [4], is more generic and has been beneficial to both the architecture research community and its practitioners [5]. This view of architecture implies a view of architecting 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 [5], [6], and it has stimulated research into re-usable architectural decisions [7], [8]. 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 [9], 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

So far (February 2011), we have trained 85 architects in our risk- and cost driven architecture approach. We are not yet at the stage where we can objectively quantify and measure the benefits mentioned above, but the feedback we are getting in conversations with the architects trained in this view is very positive, prompting us to share this approach here, together with some real-life examples the trained architects have given us.

The rest of this paper is structured as follows:

• in section II, we will show how we arrived at this point of view of what architecture is all about, and define some key concepts
• in section III, we will elaborate on the meaning of risk and cost and how they determine architectural significance
• then in section IV, 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 V, we will use three examples from industry to illustrate how this approach helps to communicate architectural significance to stakeholders
• in section VI, we will give some guidance on how to
implement this point of view
- we conclude the paper with a discussion, including some frequently asked questions about this approach

II. WHAT ARE ARCHITECTURAL DECISIONS ABOUT?

For over five years, we have been talking and writing about software architecture as a set of design decisions [4], [5]. 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 [10], 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 [11].

Let’s see if the standard definition of software architecture [10] helps: “The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution”. Going back to the example of our programmer and his daily minor design decisions, these can certainly be about the organization of the system, its components and their relationships. The programmer’s decisions may also affect the system’s evolution. There are only two concepts in the ISO42010 definition that are clearly out of the league of the programmer’s decisions: “fundamental” and “guiding principles”. These concepts can guide us towards the class of decisions architects should focus on: decisions about fundamental things and guiding principles. Ralph Johnson, quoted by Martin Fowler [11], 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.1 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 IV-B and IV-C 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 V.

A. Key concepts

“Concern” and “Decision” are key concepts throughout this paper. Concern is a well-understood, stable concept [10]: Concerns are those interests which pertain to the system’s development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders. Several models exist linking design decisions to concerns [4], [12]. These models 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 for the time being 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.

1This point of view is the basis for an architecting approach developed in Logica, a large IT services company: Risk- and Cost Driven Architecture.
III. ARCHITECTURAL SIGNIFICANCE IN TERMS OF RISK AND COST

In section II, 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 a method for 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 IV. 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 formula presented is a vehicle to explore this relationship further, as will be seen in the rest of this section.

The principle on which our method 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:

\[ AS(C) = \text{Cost}(C) + \text{Risk}(C) \]

In this formula, \( AS(C) \) represents the quantified architectural significance of concern \( C \). \( \text{Cost}(C) \) is the estimated cost of addressing the concern, as explained in section III-B. \( \text{Risk}(C) \) is the total expected cost of “things going wrong” associated with \( C \) as explained in section III-A: the cost contingency.

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 [13]. 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 [14] risk-driven model; we add cost as an equally prominent factor to consider.

A. 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 used in this paper uses the risk exposure relationship used by Boehm [15]:

\[ Risk(F) = p(F) \times I(F) \]

in which \( F \) is a particular failure scenario, \( p(F) \) is the perceived probability of \( F \) occurring, and \( I(F) \) is the estimated cost 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. For simplicity’s sake, \( I(F) \) represents all cost, technical and schedule risk 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, we must tie architectural concerns to failure scenarios. Once we have identified all independent failure scenarios related to a concern, 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 [16], which we do not have to explain here.

How do we identify failure scenarios related to an architectural concern? The key to this is in the architectural decisions. The most generic failure in solution architecting is failure to make the right architectural decisions, or put in other words: making architectural decisions that cause failure to deliver a fitting solution to the requirements. More specific failure scenarios can (and usually should) be derived from this generic scenario to determine the total risk related to an architectural decision, using standard risk evaluation calculations like [16].

If we generalize all failure scenarios related to a decision \( D \) into one scenario \( F_{D-wrong} \) in which \( D \) turns out to be a wrong decision, and consider the simplified situation in which all decisions \( D \) addressing a concern \( C \) are independent, we get:
\[ \text{Risk}(C) = \sum_{D \in DA(C)} p(F_{D \rightarrow \text{wrong}}) \times I(F_{D \rightarrow \text{wrong}}) \]

where \( DA(C) \) is the set of decisions addressing concern \( C \).

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. 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, a wrong decision could be to hard-code SQL statements to populate java objects. 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. Should this occur, the project team would have to re-factor the code, and the impact of the wrong decision ultimately translates to the total cost of this re-factor ing to the stakeholders, including the re-factor ing 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.

The above example illustrates a general point: the maximum impact of a wrong decision is the cost of reversing the decision. Apart from reversing the decision after it has proven to be wrong, there may be alternative ways of dealing with it: if the alternative ways are less costly, the impact of the wrong decision is lowered; but if they are more expensive, re-factor ing is the better choice and hence determines the impact. The upshot of this is that design decisions that are expensive to reverse tend to be more “architectural”. This resonates strongly with Klusener, Lämmel and Verhoef [17], who state that “the software architecture of deployed software is determined by those aspects that are the hardest to change.”

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 calculation method described above by making the cost estimate of dealing with wrong decisions stakeholder-dependent. One can then choose to add the costs of all stakeholders, or to filter the costs related to less critical stakeholders. In effect, 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.

B. Cost

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

\[ \text{Cost}(C) = \sum_{D \in DA(C)} \text{Cost}(D) \]

where \( DA(C) \) is the set of decisions addressing concern \( C \), and \( \text{Cost}(D) \) is the estimated cost of implementing decision \( D \). There are many documented ways to estimate cost in software engineering (e.g. [18]), which we will not discuss here. Whatever method is chosen, risk factors must be excluded from this cost estimate, to prevent double-counting risks into the architectural significance function \( AS(C) \) above.

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.

IV. 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 [19]. 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.

A. 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, defined in [10] as “those interests which pertain to the system’s development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders.” Viewing architecting as a risk- and cost management discipline sheds light on what is “critical or otherwise important”: this subjective qualification can be replaced by the more objective “impact in terms of risk and cost”. This replacement 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 [19], the ASR definition is borrowed from [20]: “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.

B. Architecting workflow

In [19], 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 III, 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.” [19]. 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 ± 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.

C. 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 III-A 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 the second factor is often equal to 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 blue line, the impact of D being wrong by a red line, and the moment tD of committing to decision D as a vertical yellow line:

- after the moment of decision, the cost of reversing an architectural decision, and thus 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 blue 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 \( P(D \to wrong) \) decreases, and the blue 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. \( R(D) = P(D \to wrong) \times I(D \to wrong) \) may either decrease or increase.
- If the emerging information points more and more in the direction of D having been wrong, both \( P(D \to wrong) \) and \( I(D \to wrong) \) 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 section III-A 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 [21] using decision trees and real-option theory.

V. 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 [22]. 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. In section V-A, 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 [23] architects to projects or product organizations.

A. Examples from practicing architects

In this section, we will present some examples from real projects\(^2\), 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

\(^2\) All examples are from real projects; due to company confidentiality constraints, the examples have been abstracted away from their specific project context.
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.

VI. 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 [14], [24].

VII. 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.


A. Related work

Most literature relating risk and software development 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. 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?' [25]. Another class of papers discusses sophisticated techniques for computationally handling risks, using Bayesian networks, fuzzy set theory, and the like; [26] is an example hereof. A third type of articles focuses on conceptual models for handling risk in software development. A seminal paper in this category is [15].

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) [27] is a list of risks and non-risks. By studying the output of a series of such ATAM evaluations, Bass, Nord, Wood and Zubrow [28] were able to reveal and analyze risk themes specifically geared towards software architecture. Slyngstad et al. [29] 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 a low level of awareness that lack of software architecture evaluation is itself a potential risk.

Viewing risk and cost as drivers in architecture decision making has led to approaches like the Cost Benefit Analysis Method (CBAM) [9] and the Architecture Rationalization Method (ARM) [30] that relate architectural decisions to the benefits they bring to an organization, studies that emphasize business implications of architectural decisions [22], and approaches that consider architectural decisions as investment decisions [21], [31].

Fairbanks [14] 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.

B. 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.

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. [17], 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 section III-A. Slyngstad et al. [29] have surveyed risks in software architecture evolution, and present the most common risks for architectures of existing systems and their associated mitigation activities.

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 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 an invitation to tender (ITT): the ITT 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 ITT 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) 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.

2) When the solution architect is involved in the analysis work. A good example of this is the Cost Benefit Anal-
ysis Method [9], a method for performing economic modeling of software systems, centered on an analysis of their architecture.

It is known from many experiments in decision making 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 [32]. 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.

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.

4) Does this also apply to {infrastructure||business||systems||...}-architecture?: Of the 85 architects we have taught 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.

In conclusion, even though it is derived from practices and research related to Software Architecture, the risk- and cost driven view of architecture is applicable to the wider area of Solution Architecture.

C. Conclusion

We have presented and elaborated a view of architecting as a risk- and cost management discipline. The view goes beyond software architecture alone: it includes other architecture genres that can be captured under the name Solution Architecture.

The risk- and cost driven view on architecture is part of a solution architecture training programme that has so far been taught to 85 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 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.

In our future work, we plan to further analyze the types of concerns practicing architects spend their time on, and the impact of available information (such as expected value) on decisions architects make.

REFERENCES


