

Chapter 9

Software Architecture Design Reasoning

Antony Tang and Hans van Vliet

Abstract Despite recent advancements in software architecture knowledge management and design rationale modeling, industrial practice is behind in adopting these methods. The lack of empirical proofs and the lack of a practical process that can be easily incorporated by practitioners are some of the hindrance for adoptions. In particular, the process to support systematic design reasoning is not available. To rectify this issue, we propose a design reasoning process to help architects cope with an architectural design environment where design concerns are cross-cutting and diversified. We use an industrial case study to validate that the design reasoning process can help improve the quality of software architecture design. The results have indicated that associating design concerns and identifying design options are important steps in design reasoning.

9.1 Introduction

Software architects make a series of design decisions when designing system architecture. Despite the need to make correct architectural design decisions, architects often omit to provide design rationale or justifications for their design decisions. Software architects instead focus on creating design outcomes. Presently there are no commonly accepted practices in the industry to carry out systematic design reasoning, architects often rely on their experience and intuition when making design decisions. Such an unstructured decision making approach has certain implications on design quality, experienced architects are more likely to make better design decisions. On the other hand, inexperienced designers may not design as well. This case

Antony Tang (✉)
Swinburne University of Technology, Melbourne, Australia, e-mail: atang@swin.edu.au

Hans van Vliet
VU University Amsterdam, The Netherlands, e-mail: hans@cs.vu.nl

study illustrates a methodology to overcome the ad hoc practice of architectural design decision making and to improve the quality of software architecture design.

In general, practitioners learn software architecture design principles through textbooks and formal training, exercising these design principles in practice often requires knowledge and experience that is mostly learned on the job. Software design knowledge and experience are difficult to teach and articulate. Recent studies have shown that such knowledge is design rationale related [48, 71]. Design rationale can be characterized in two ways (1) they are the reasons for making a decision and choosing a solution, and (2) explaining the relationships between a solution and the context of that solution. By externalizing design reasoning, it is aimed to improve decision making and to capture documented evidence for design verification and system maintenance.

In this chapter we explain the basic elements in architectural design reasoning. We demonstrate the modeling of these elements in a UML-based model called Architecture Rationale and Elements Linkage (AREL) in Sect. 9.3. In Sect. 9.4, we describe a design reasoning process to support software architectural design activities. In Sect. 9.5, we describe an industrial case that demonstrates how a reasoning process can improve the quality of an architectural design.

9.2 Software Architecture Design Reasoning

In software architecture design, architects apply cognitive reasoning even though they may not think about it consciously. An understanding of such a reasoning process can be quite helpful to delivering good design. Designers' judgment can be inadvertently biased due to personal preferences and past experiences. This situation is quite common and it may have an adverse effect on the quality of a design. However, many systems have been built quite successfully without the explicit employment of design reasoning methods, why? This may be due to the involvement of experienced people. Successful projects often rely on people with experience and good judgment. Some IT professionals seem to have an uncanny way of foreseeing problems, formulating solutions and making just the right decisions consistently. On the contrary, there are practitioners who design poorly, they over-engineer a solution, underestimate the effort, miss out key requirements, select the wrong technologies and deliver poor quality design. The challenge is how to systematically improve the reasoning abilities of designers to consistently deliver a satisfactory design, and to improve the quality-assurance process of architecture design.

In a study that examines if design reasoning techniques make any difference to design quality [318], test participants in the experiment were given a simple reasoning process, they were asked to use this process and verbally explain their design reasoning as they designed. As a result they generally produced a higher quality design, especially those who were less experienced could produce designs similar to those by experienced designers. The study shows that those who were required to externalize their design reasoning were probably more careful and methodical in

designing their solutions. This contrasts with the participants of the control group who largely used intuition and knowledge to design. The control group's objective was to complete the design and satisfy the requirements without having to justify them. Other studies have also shown that design rationale documentation helps designers understand and reason about their design [51, 171].

A well-designed architecture should be justifiable through sound and logical design rationale. The design reasoning process ought to consider all relevant architecture requirements, address the design issues, consider trade-offs between the design options before deciding on the outcomes. The explicit representation of this tacit knowledge serves many purposes in the development life cycle such as review and maintenance. Table 9.1 summarizes the purposes for having such a rationale-based architectural design approach.

Back in 1958, it was suggested that argumentation could be used to induce conclusions from contextual data [323]. This approach explicitly represents the design deliberation process. One of the characteristics is that they model as links the relationships between design goals and design results. Examples include Issue-Based Information System (IBIS), Decision Representation Language (DRL), and Questions, Options and Criteria (QOC) [228]. Unfortunately, these methods have not been successful in practice because of their difficulties in capturing and communicating design rationale [298, 200].

Another approach is to use templates to aid design reasoning capture, including [71] and [325]. Such an approach is beginning to receive attention in the industry as practitioners recognize the importance of recording design reasoning. UML CASE tools such as Rational Rose and Enterprise Architect¹ provide some easy-to-use facilities to capture design rationale. Although template-based methods can capture design rationale, they provide limited support to relate the contexts of design decisions.

In order to make design reasoning easy to adopt by software development organizations without losing the design reasoning capabilities of argumentation-based methods, we have developed AREL as a hybrid approach that incorporates design rationale template and design reasoning relationships based on these previous works [71, 228, 325]. AREL has been designed to capture useful design reasoning information with minimal documentation overheads.

9.3 Modeling Architecture Design Reasoning

What exactly is design reasoning? Is it a reason for having a system or is it some justifications on how a system is designed? First of all, let us consider a simple reasoning model that comprises of three elements: *inputs–decisions–outputs*. The *inputs* are the requirements and goals that need to be met by a system; the *decisions* are the decisions made in designing the system; the *outputs* are the results of

¹ Design rationale support in Enterprise Architect is implemented using the AREL plug-in tools.

Table 9.1 Purposes for having a rationale-based architectural design approach

<i>Support Software Architecture Design</i>
<i>Deliberating and Negotiating Design</i> – design rationale allows designers to systematically clarify the issues and possible solutions, and to evaluate decisions against well-defined criteria. As such, it allows designers and stakeholders to deliberate and negotiate a solution
<i>Justifying Design Decisions</i> – design rationale can explicate tacit assumptions, clarify dependencies and constraints, and help justify why a particular choice is selected from amongst the alternatives
<i>Applying Trade-off Analysis</i> – a design decision often involves resolving conflicting requirements that cannot be fully satisfied simultaneously. When trade-off analysis method such as ATAM [34] is applied, the prioritized requirements and utility tree form the reasoning of the compromised decision
<i>Structured Design Process</i> – design rationale supports a structured and accountable design practice. It provides a pertinent understanding of the context, the stakeholders, the technologies and the situations in a project
<i>Design Validation</i> – design rationale explains why certain design decisions have been made, and provides the necessary information for independent architecture design validation and review
<i>Communication and Knowledge Transfer</i> – design rationale can help business analysts evaluate conflicting requirements and new designers learn the architecture design
<i>Support Maintenance Activities</i>
<i>Retaining Knowledge</i> – if system maintainers are not the same people who originally developed the system and the design rationale is not available, maintainers would have to second-guess the intangible rationale
<i>Understanding Previous Design Alternatives</i> – design alternatives can help maintainers appreciate what choices had been considered in a decision. They help maintainers to understand design options that were considered unviable or allow them to consider an alternative that was not viable at the time but can now be used
<i>Understanding Design Dependency</i> – design decisions can be interdependent and cut across a number of issues. Changing a decision may have ripple effects on other parts of a system. Recording design rationale and their interdependency helps alleviate the concern of overlooking related issues
<i>Improving Maintenance Productivity</i> – in an experiment, it was shown that a group of designers equipped with the design rationale can work faster and identify more changes that are required in a system maintenance exercise than a control group without the design rationale [51]
<i>Predicting Change Impact</i> – design rationale could assist maintainers to predict which part of the system is subject to consequential change [317]
<i>Providing Traceability</i> – maintainers would be able to trace how design artifacts satisfy requirements in a system with some explanation [259]

the design. Without the inputs, we would miss out on the contextual information that tells us why we need the design. Without the design decisions, we may not understand the justifications or reasons for choosing a design. Therefore, in modeling design reasoning, we need to depict the causal relationships between the design inputs, design decisions and design outputs. We suggest that this relationship is a simple causal relationship between the causes of a design and the effects of a design.

In this way, *design reasoning* is modeled by a description of the design context, the design justification, the design outcome and their causal relationships.

IEEE-1471-2000 specifies that architectural rationale should provide the evidence for the consideration of alternative architectural concepts and the rationale for the choices made. It is a general guideline that does not provide much detail to help implement a design reasoning model. In a discussion session on updating IEEE-1471, some refinements have been suggested to overcome this issue [23].

AREL captures both the design rationale and the design model, which is realized through a UML tool, Enterprise Architect (EA). The design model includes architecturally significant requirements and the actual architecture design. AREL supports the association between the design model and the design rationale of its design decisions. The AREL model is based on the reasoning model described earlier: (a) design concerns raise a design decision, (b) design decision captures design issues and design rationale; (c) the design outcomes are the results of a design decision.

Figure 9.1 shows the conceptual model of AREL. Design concerns such as requirements cause the need for design decisions to be made. When a design decision is made, it is justified by its design rationale. To make a design decision, different design options may be considered, these alternative designs can help architects consider their relative pros and cons. When a design decision is finally made, there would be a chosen design and may be some alternative designs. Alternative designs are those design options that have been considered but discarded. Alternative designs are important because they are evidence to show that the designers have considered more than one design options before making a design decision, they also show the reasons why these alternatives are not as appropriate as the chosen design.

A chosen design element is a UML entity to model the design, e.g. class, component or database table. Each design element can influence other parts of a system due to its behavior or constraints. These design elements will in turn become design concerns that influence other parts of a design. For instance, using AJAX to construct web pages creates a consequential issue of having to handle the web browser’s BACK button. AJAX being part of the solution space therefore becomes a part of the problem space, i.e. a new design concern. Such design consequences could trigger a chain of interdependent designs and design decisions.

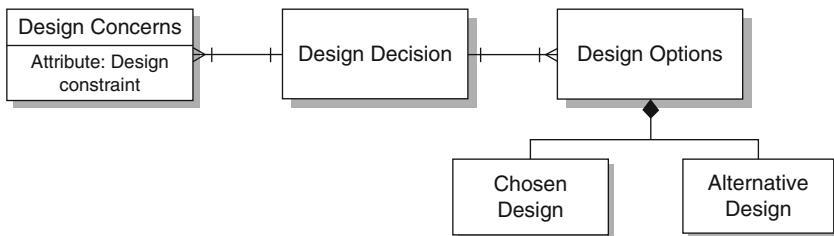


Fig. 9.1 A conceptual model of design reasoning in AREL

Table 9.2 Types of design concern

<i>Purposes and goals</i> – the business goals of a system
<i>Functional Requirements</i> – functional goals of a system
<i>Non-Functional Requirements</i> – quality attributes that a system must fulfill, e.g. performance and usability
<i>Business Environment</i> – organization and business environmental factors that influence architecture design, e.g. long-term or strategic organization goals
<i>Information System (IS) Environment</i> – environmental factors that influence the construction and implementation of the system, e.g. budget, schedule and expertise
<i>Technology (IT) Environment</i> – technological factors that influence the architecture, e.g. current organizational technologies and policies
<i>Design</i> – a chosen design (outcome) has some influence on the rest of the architecture, e.g. the selection of an operating system constrains the choice of the development tools

9.3.1 Design Concern

Capturing the causes of a design decision is vital to comprehending the reasons of a design. The inputs that are the causes or motivations of a decision are *design concerns*. Design concerns represent the context of a design. They can be anything that influences the design decision. Functional and non-functional requirements are examples of design concerns. There are, however, many design concerns that are often omitted even though they play a significant role in software architecture.

There are different types of design concerns (see Table 9.2) and they influence decisions in different ways. *Requirements* drive and motivate the creation of designs. *Purposes and goals* provide a context to guide the design. *Environmental* factors constrain the available choices of an architecture design. In a case where an architect designs a B2B website, examples of the environmental design concerns could be (a) business environment – outsourcing of a system has implications on the maintainability and support requirements; (b) IS environment – time to market is 3 months; (c) IT environment – organisation standards to use ASP.Net and Oracle.

These design concerns exert constraints or influence on design decisions and the eventual architectural design. A design constraint is a limiting factor which specifies the conditions that a viable design solution must fulfill, e.g. the performance of a database engine. As design constraints are important to design decisions, architects must recognise them and ensure that they can be satisfied by the architecture design.

9.3.2 Design Decision

Design decisions can sometimes be made without applying any systematic reasoning or documenting their justifications. Making appropriate design decisions by intuition relies on the abilities and experience of the designer; making design decisions through systematic reasoning, on the other hand, requires explicit justification using design rationale. Documenting design rationale for the entire system can be

Table 9.3 Architecture design rationale

<i>Qualitative design rationale</i>
<i>Design Issue</i> – the issue to be dealt with in a decision
<i>Design Assumptions</i> – document the assumptions that are made in a decision
<i>Design Constraints</i> – document the constraints on a decision that can be of a technical or contextual nature
<i>Strengths and Weaknesses</i> – state the strengths or weaknesses of a design option
<i>Trade-offs</i> – document a balanced analysis of what is an appropriate option after prioritizing and weighing different design options
<i>Risks and Non-risks</i> – document the considerations about the uncertainties or certainties of a design option
<i>Quantitative design rationale</i>
<i>Cost</i> – quantifies the relative cost in areas such as development efforts, platform support, maintenance cost and other intangible costs such as potential legal liabilities
<i>Benefit</i> – quantifies how well a design option may satisfy the requirements and the quality attributes
<i>Implementation Risk</i> – represents the risk that a development team may not implement the design successfully due to reasons such as the lack of capability or experience
<i>Outcome Certainty Risk</i> – represents the risk that a design may not satisfy the requirements because they are technically unachievable or not well-defined

very costly and probably unnecessary, and so we focus on architectural issues that are often complex, intertwining and require much investigation.

Design rationale is the reasons for choosing a particular design from a range of alternatives at a decision point. To do so, we must first articulate the design issues to resolve. An issue may simply be designing to satisfy some design concerns. For instance, what data do I need to show in the user interface? More often than not in architectural design, issues are more complicated because of conflicting and competing influences from different design concerns such as quality requirements. For instance, how do I retrieve the data securely and maintain system performance? There may be more than one possible solution. Design rationale therefore helps the reasoning process and captures the justifications for selecting a solution. It explains why a design is better than the other alternatives. AREL uses qualitative and quantitative design rationale to capture such justifications (see Table 9.3).

Qualitative design rationale supports design reasoning by way of arguments. Architects may document the justifications of a decision by arguing the relative advantages and disadvantages of different design options, or using a trade-off analysis method such as ATAM [34]. Quantitative design rationale records the relative costs, benefits and risks of a design option using ordinal numbers, between 1 (the lowest) and 10 (the highest). The main reasons for capturing quantitative rationale are to allow a quantifiable comparison between alternative design options, and enable architects to highlight risky or costly decisions that need to be investigated further. For example, if the implementation risk is high, architects may continue to decompose the architecture through a series of decisions until the risk becomes manageable [315].

9.3.3 *Design Outcome*

The result of a design decision would be some chosen designs that are a part of the total solution. This chosen design either realizes the requirements of a system, or it provides some design structures that are used in realizing the requirements. The design outcomes can be any design artifacts, example are architectural model, database models, design components and classes.

A chosen design outcome may influence another part of a solution and as such it becomes a design concern to a new decision, as we have seen in the AJAX example earlier. Since a design outcome can associate with other decisions through the causal relationship, it is possible to have a graph that consists of a chain of relationships that connect design concerns, decisions and design outcomes.

9.4 An Architectural Design Reasoning Process

AREL is a model which defines the information used in design reasoning, it underpins the software architecture design reasoning process described here. Design methods such as the waterfall method, iterative method or agile method focus on the organization of events and teams in software development; technology based methods such as object-oriented analysis and design focus on modeling techniques that produce specific design artifacts. A reasoning based design method, however, has a different perspective – the focus is on using reasoning to create and justify design decisions made for architectural design. The considerations used in a design reasoning approach are therefore broader and not just focusing on producing design artifacts. The design reasoning approach is not meant to replace other design methods, but rather it adds a new dimension to designing software architecture to complement existing design methods.

The architectural life-cycle described by [146] comprises three distinct activities: architectural analysis, architectural synthesis and architectural evaluation. Architectural synthesis is the design activity in this architectural life-cycle. The design reasoning process described in this chapter primarily addresses the architectural synthesis activity but it also covers architectural analysis and evaluation. The shaded area in Fig. 9.2a shows the scope of the design reasoning process with respect to the architectural activities described in [146]. The architectural design reasoning process spans across architectural synthesis and the other activities because design reasoning would involve analysis as well as evaluation.

Chapter 1 describes architectural design as a problem-solving activity, the design workflow consists of three activities: requirements gathering, backlog creation and design evaluation. Creating backlog is an important activity in which architects articulate what design problems are there to be solved. It requires design context and requirements as inputs for defining the design problems in the backlog. All three activities require design reasoning support (shaded area in Fig. 9.2b).

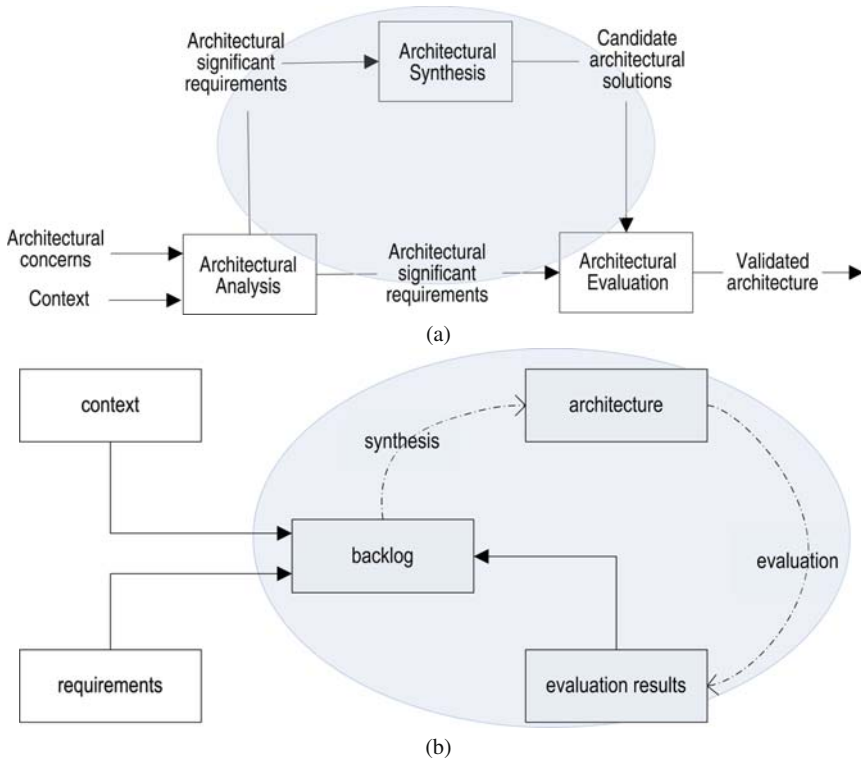


Fig. 9.2 (a) and (b) Design reasoning coverage at two levels of architectural design

Design is a cognitive activity that requires the architects to organize, induce and assimilate different kinds of information. Early work has indicated that this cognitive process can differ between architects where experienced architects generally use a better design approach, and therefore yielding better results [318]. As such, it would be advantageous to investigate into design reasoning strategies, making them explicit so that there is a systematic approach to considering design problems. A conceptual model for design decision has been suggested by [162] where they describe the architecting process, however key activities that are required to support this model have not been articulated.

In order to address the design reasoning gap in software architecture, our starting point is to make use of the causal relationships between design concerns, design decisions and design outcomes. We suggest to take five steps in the design reasoning process. The five steps (i.e. step 1–5) are depicted by the numbered arrows between the entities (Fig. 9.3). The arrows indicate the design reasoning steps, and they can be repeated for a design. By connecting AREL entities that are causally related, designers can trace the causes and effects of those design decisions. These 5 design reasoning steps are performed repeatedly to decompose the design and create design details, shown by the layers of decisions in Fig. 9.3.

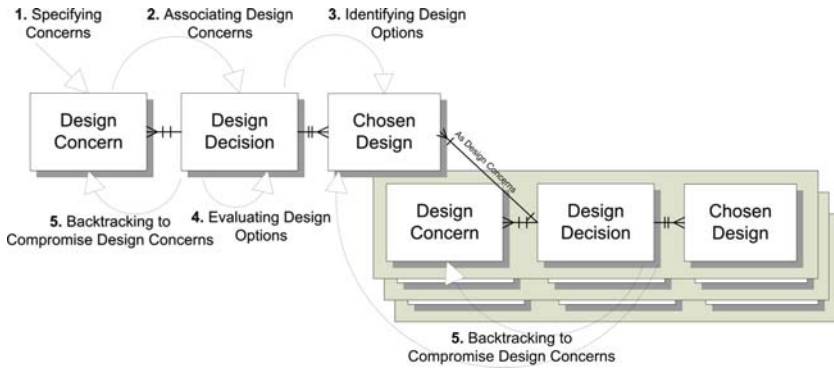


Fig. 9.3 A design reasoning process based on AREL

The results of the design reasoning process are captured, i.e. design decisions, design rationale, design outcomes and their relationships. Figure 9.3 shows the relationships between these entities in an AREL model.

1. *Specifying Design Concerns.* Business goals and Architectural Significant Requirements (ASRs) are the basic inputs, we call them design concerns, to the design reasoning process. We assume they are available to start the design process. The elicitation and analysis of business goals and ASRs are the initial step which analysts and designers achieve by using requirements engineering methods [235, 142] and architectural analysis methods [176, 172].

Business goals and ASRs themselves reflect important architectural decisions that have been made and these decisions would influence subsequent architecture design [46], so should a design reasoning process be extended to deal with ASRs? A clear delineation between requirement analysis and design can be difficult to define because they influence each other mutually. At a broad level, specifying design concerns are different from synthesizing a design solution. However, as seen in some architecture analysis methods [176, 173], architectural design can compromise ASRs in order to create a workable solution. This process is described in step 5.

2. *Associating Design Concerns.* Design decisions are made because architects need to work out how to design for particular situations that arise from a combination of design concerns. The idea is to consider relevant design concerns in conjunction and finding a solution for them. This concept is similar to the ideas of creating design topics in [44] or populating the backlog (Chap. 1). For instance, to make a decision on how to design an authentication server, two design concerns, i.e. security requirement and performance level, must be considered. If performance as a design concern is not associated to this decision, one may end up having a design that addresses the security requirement with poor performance.

This idea of grouping design concerns and relating them to a decision bears similarities to eliciting use case scenarios where multiple requirements are involved

[172], or eliciting architectural issues for design synthesis [85]. The graphical modeling of the causal relationships between design concerns and design decisions in AREL enables architects to visually relate them for inspection and traceability. Current practice in the industry requires an architect to have the knowledge and experience to associate related design concerns, and they often do that from a set of textual specifications. It is therefore easy to omit a design issue where interrelated design concerns should be considered in a design scenario.

How does one know that certain design concerns must be considered together to form a design topic? This is essentially asking how an architect knows what to design. There are two possibilities: Firstly, a software architect would have certain basic knowledge and understanding of the design principles for associating related design concerns; secondly, architectural design knowledge can assist architects by showing general design concern associations with design topics.

A design decision can trigger a chain of subsequent decisions, and typically it is for deciding how to further design the system. For instance, the layering of software components might be a high-level decision to group related components together, and follow-on decisions are required to define common interfaces between software layers.

So how deep should the design reasoning process go before an architect is satisfied? This is a question related to how much details is enough in an architectural design to enable detailed design and development. We address this issue by employing a risk mitigation strategy. Architects are required to estimate the risk of implementing a design. If the risk is low, it means that the architect is confident that the design contains sufficient details, otherwise the architect needs to further explore the design [315].

3. *Identifying Design Options.* An important aspect of design reasoning is to identify possible design options. This is a key step in synthesizing a design. It requires an architect to have sufficient knowledge of the problem and certain creativity in deriving potential solutions. This can come from understanding the relevant first principles of design, having some design experience and so on.

In a study [318], it has been observed that architects and designers tend to bias towards the first impression of a design. For participants who were involved in a test procedure to use a design reasoning process, it has been observed that they would modify their initial design after they consider alternative design options. On the other hand, designers who do not use a design reasoning process, especially for the inexperienced designers, the first impression of a design usually becomes the final design. This observation implies that designers who can identify and consider alternative design options can improve the design quality.

4. *Evaluating Design Options.* At each decision point, an architect would need to choose which identified design options is best suited to meet all the related design concerns and fulfilling the design constraints exerted by these design concerns. There are a number of possible results from an evaluation: (a) there is no design solution that can meet the relevant design concerns; (b) there is a single possible solution; and (c) there are more than one possible design solutions. In case of (a), certain trade-offs may have to take place to relax the constraints of the decision,

different trade-off analysis methods could be employed [176, 3]. In case of (c), an architect has to evaluate the pros and cons of the available options to choose the best option.

In analyzing the pros and cons of different design options, a qualitative reasoning approach may work quite well. The approach proposed in [325] employs different types of design rationale as a guideline for architects to assess the weakness, benefits and other aspect of a design.

5. *Backtracking Decisions to Revise Design Concerns.* Design decisions are often interrelated, a software design decision often leads to a chain of subsequent design decisions. Subsequent design decisions have to be made because (a) the initial design decision lacks design details and thus requires further elaborations; (b) the initial design decision has created new design issues that need to be resolved. As interrelated design decisions are made, architects may find that the eventual solution is not viable. Therefore previous design decisions have to be revised and new decisions to be considered.

When no design solutions can be found at a decision point, the design concerns that dictate the decision must be re-examined because of their causal relationships. Each design concern constrains a design decision in some ways, when the constraints cannot be satisfied by any solutions, then the decision is not viable. A compromise can be reached if some of these constraints are relaxed. To do so, architects can backtrack design decisions to relax their constraints.

If a design constraint comes from a requirement, architects would have to negotiate with the relevant stakeholders in order to have them relaxed. If a design constraint to be relaxed is itself a design outcome, then changing the design outcome implies reconsidering all previous decisions that have led to this design outcome (see step 5 in Fig. 9.3). In this case, design reasoning backtracking involves revisiting previous decisions and design constraints to ensure that the design constraints of all related design concerns can be fulfilled.

The design reasoning process focuses on how to reason with a design. The key steps in the design reasoning process are associating design concerns to decision points, justifying each decision point, checking that all design constraints are met, and backtracking to revise decisions if no viable solution is found. These simple steps can be used in conjunction with other design methodologies.

9.5 Applying AREL to an Industrial Case Study

In an industrial case study, we apply the AREL model to a software development project to see if a design reasoning process can improve design quality in comparison with the conventional design process. A consulting and software development firm had a contract to supply a document management system to a large company in Australia. The system will provide document repository functionality, document classification, knowledge search, workflow, single-sign-on and ubiquitous access facilities to this engineering firm. The design and implementation is phased, and in this

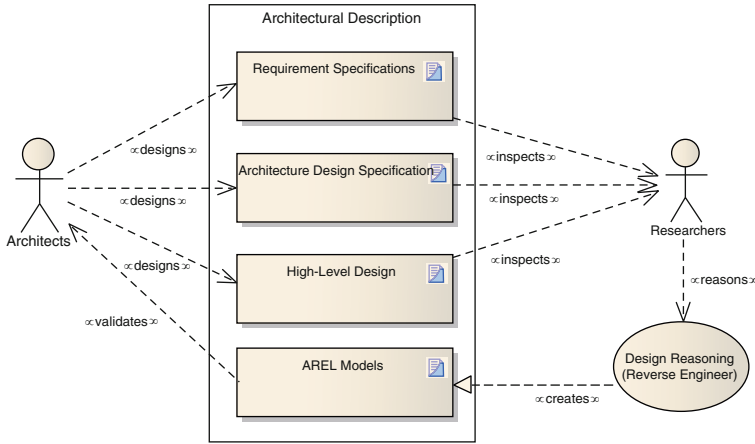


Fig. 9.4 Research approach used in the case study

case study the architectural design for the system infrastructure and two application systems are studied.

The consulting firm has a *design team* formed by architects and designers to carry out requirement elicitation, architectural and software design. There were four architects/designers in the design team. All of them have had many years of experience in the IT industry and at least 5 yrs. on document management applications. The firm uses an internal development and documentation standard, and they also have an internal and external review process. The development methodology can be typified as structured analysis and design. The design team does not use any design reasoning methods during design.

The *research team* consists of two researchers who work independently of the design team to analyze the information. The research team analyzes the specifications prepared by the design team, these specifications include functional requirement specification, architectural design specification, high-level design specification and traceability matrix between requirements and designs. With the supplied documentation, the research team constructed the AREL models using the design reasoning process to reverse engineer the design decisions made. The design team then validate the issues and questions raised by the design reasoning process with the design team.

9.5.1 Analyze the Design by Reasoning

The researchers first examine the specifications, it has been observed that there is very little documented design rationale. When they exist, they are buried within the text of the specifications. The researchers then apply the reasoning process to reverse engineer the design decisions. The researchers imported the summarized specifications into an AREL model. This was achieved by a custom developed software to

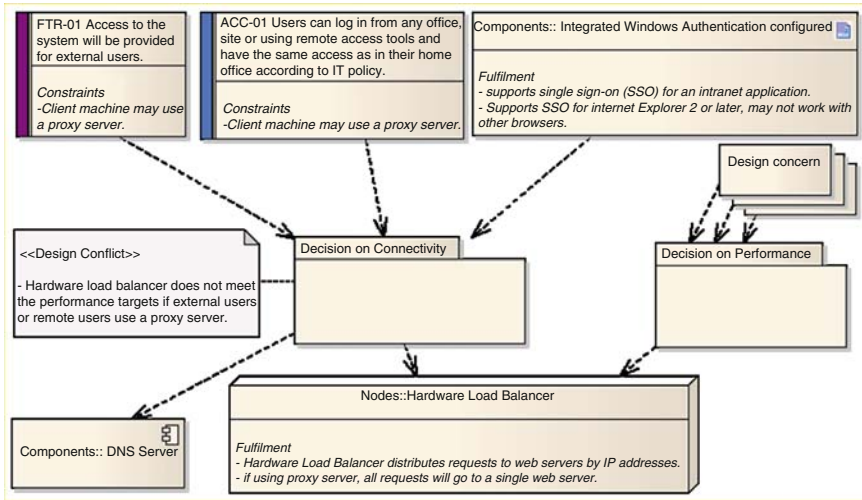


Fig. 9.5 An example AREL decision diagram

scan the specifications and retrieve the requirements. The import process created the *design concerns* and *chosen designs* as UML artifacts stereotyped by `<<architectural elements>>`. The nodes in the model contain a unique identity and a brief description of a requirement or a design component. There are a total of 419 design concerns, consisting of 254 functional requirements, 77 non-functional requirements and 88 pieces of contextual information about the system. All of them were imported from the specifications into the model. There are also a total of 86 design components.

Using the AREL model, the researchers carried out a reverse engineering exercise to discover the design decisions. This is done by *associating* relevant design concerns to a design decision (using reasoning step 2), and then find all the possible design outcomes that are affected by this decision (using reasoning step 3). Using this process, the researchers hope to uncover the design reasoning and find any design issues. The result was a series of AREL decision diagrams, an example is shown in Fig. 9.5.

The example in Fig. 9.5 shows the requirements related to secured access to the document management system. The researchers have associated the relevant requirements to a design decision node (i.e. Decision on Connectivity). The design concerns are: (FTR-01) access to the system will be provided to external users; (ACC-01) users can log in from any office or from home using remote access tools; Component to support integrated Windows Authentication Server; and performance requirements. The decision to consider connectivity is to employ a hardware load balancer to realize the design and to use a DNS server.

It was indicated in the specification that the hardware load balancer cannot support external proxy server connections because web accesses cannot be routed to the right server if proxies are used. The design to use a load balancer is primarily for the performance requirement. The design outcome obviously contradicted with the

requirements where external users most likely would access through a proxy server. This fundamental requirement cannot be satisfied by the existing hardware. Moreover, if the designers attempt to implement this unwittingly, the performance of the web site would be adversely affected.

By associating the relevant requirements to the design outcomes, we can identify the contradictions in the design because not all the design concerns can be fulfilled by the design. Furthermore, inter-related design may conflict each other. If the load balancer does not support connections through the proxy servers, then there is no way to support external and internal users who access the system from home. This design conflict was raised with the design team and the design team acknowledged this oversight because they had not associated the external access requirement to the load balancer design.

9.5.2 Applying Design Reasoning in the Case Study

The researchers analyzed the system by reverse engineering the design decisions made to create the software architecture. Many cases of ambiguous design rationale were identified. These cases were then presented to the design team to validate if they were real design issues, and all the findings were confirmed to be valid. Through this exercise, it was shown that a systematic design reasoning process would help architects to uncover design issues and achieve a better quality architectural design.

Using the design reasoning approach, researchers have identified 83 issues, of which 29 are ambiguous design concerns, eight issues related to designs that cannot be fully justified through reasoning with the design concerns, and 46 issues that are ambiguous description of design outcomes. With the identified design issues, we analyze the likely causes of why they occur so that we have some insights on how design reasoning may help to improve the situation. The following are cases where design reasoning has uncovered architectural design issues. These cases represent failures that could be avoided if design reasoning steps are taken.

1. *Missing cross-cutting design* Design concerns can cut-across different parts of the software architecture. In the case study, there are many examples of not considering cross-cutting design concerns. These cross-cutting design concerns need to be considered or associated together when designing because they affect each other. An example from the case study is the missing association between user authentication of internal and external users. Currently the company employs an Active Directory to authenticate intranet users, this mechanism supports single sign-on for software applications within the company. Single sign-on is a company policy. However, the architects have omitted to associate this mechanism to another future requirement (FUT-004-01), i.e. supporting external users' login. When these two requirements are associated together, a new significant architectural issue arises – “how to authenticate external users?”.

When the architects were interviewed, they indicated that they had not really considered this particular aspect. When the researchers discussed the possible impacts of this omission, the architects then started to identify potential design issues that can arise from it. There are two new design issues concerning security policies, redefining user group privileges and access rights, and access control of external users. The architects agreed that these newly discovered design issues should be addressed. This example illustrates that missing associations of cross-cutting design concerns can cause architectural significant design decisions to be omitted. The likely reasons for such omission are because the system is inherently complex, and system analysis and design is based on textual specifications where minor details can be overlooked. Design reasoning step 2 (see Sect. 9.4) can circumvent this problem by encouraging architects to consciously associate related design concerns to identify new design issues.

2. *Conflicting design concerns.* When the researchers map the design concerns or requirements to the AREL model, it has been found that some design concerns cannot be realized by the current design because they conflict with each other. Architects did not realize that interrelated design concerns should be dealt with together, resulting in hidden requirement conflicts that are not detected by the architects.

For instance, one requirement is to allow users to ubiquitously access the system and retrieve all types of documents using remote access tools (ACC-002-01); another requirement specifies that all files are to be converted to readable format for viewing when the user does not have the program installed (EDM-008-01); the system design specifies that remote tools such as Citrix and blackberries are to be used (AA-ASM-908). These requirements and design come from different areas of the specifications. When they are analyzed together, conflict arises: (a) requirement ACC-002-01 is by itself ambiguous, it is not specific as to what remote tools should be included in the supported list of tools. This is somewhat clarified by the design assumption AA-ASM-908; (b) when blackberries are required to render documents for viewing, a design conflict is detected. The blackberry device cannot render documents for viewing.

In this example, the conflict was detected when relevant design outcomes and requirements are linked together. The design issue becomes visible and when it is realized that no design option can satisfy the combined requirements, a design conflict is thus detected.

3. *Ambiguous design concerns.* When the researchers map the design concerns, in terms of requirements, to the design components using the AREL model, the researchers found some inconsistencies between the design concerns and the design outcomes. Most notably, there were design components that could not be traced to the specific design concerns. For instance, the architectural design specified the use of the load balancer, the idea was to distribute web requests to multiple web servers. However, there were no specific non-functional requirements that outline what level of performance was required. There were, however, general requirements which state that the system must have “improved client performance” and “automated failover”. The researchers verified

that these design concerns were the ones that had driven the design decision on performance.

The architects were asked about this decision, and the architects thought that it would be a good idea to have a load balancer. The architects might be right intuitively with regards to improving the general performance of a system. However, the decision from this ambiguous design concern had created two architectural issues: firstly, what level of performance should the system deliver? Secondly, the use of a load balancer has created new design issues relating to preserving stateful user sessions in the application design. Subsequent to this realization, the architects have backtrack the decision and revisited this design area (step 5 of the design reasoning process).

4. *No apparent reasons for a design decision.* Even though the requirement and design specifications of the system were organized and extensive, there was very little design rationale that was explicitly documented in these documents. After reconstructing the AREL design reasoning map, it was discovered that in some of the designs, the design reasoning could not be deduced by the researchers. For instance, requirements PRO-007-01 and PRO-007-02 specified that documents that are stored in the application system would be reviewed periodically. The architects chose to realize these requirements by using a standard reporting module that require the document reviewers to search for the documents available for review.

When the researchers inquired what other design options had been considered, it was found that the first solution that came to mind (i.e. standard reporting module) was the final solution. There was no evaluation of alternative design solutions. One of the possible design solutions in this case was to create an event driven reporting mechanism where reviewers are notified automatically when documents are due for reviews. If a design reasoning approach was used, the architects might have considered this alternate solution. Another study [318] has shown that designers who do not employ design reasoning can fail to consider alternative design options.

In summary, it has been found that the design process undertaken by the architects were inadequate to address all the architectural design issues of the system. We suggest that this is due to the lack of a systematic design reasoning process. Architects' design analysis are functionality focused, and this approach seems to be ineffective when dealing with cross-cutting concerns in architectural design. A design reasoning approach offers a new perspective to systematic architectural design.

9.5.3 Other Findings

Interviews were conducted with the design team at the end of each review session. They were asked to comment on the research findings and the methodologies, the following comments were given by the architects and the designers:

1. *Graphical communication tool.* The AREL model can be a useful tool to communicate design reasoning with their clients because it is easy to trace the graphs. Architects want to use them to validate requirements and architectural design reasoning with their clients. They suggested to have the clients sign-off the design reasoning model, and use it to manage any potential changes to the design reasoning.
2. *Facilitate negotiation.* The AREL model can depict line of reasoning to the client, including the viable options and their pros and cons. It allows the software architects to argue the necessity to compromise certain requirements. Thus it becomes a reasoning tool to help discussions and deliberations of requirements and design.
3. *Traceability.* Supporting tools are available to import specifications to create AREL models, making it easier to build the AREL model. This facility enables the software architects to build models for visualization and better traceability. The requirement traceability matrix that is developed by the design team currently does not completely document the relationships between requirement and design, the AREL model has provided a better traceability in this case.
4. *Facilitate learning.* The information contained in the UML diagrams enables architects in other areas of design to quickly understand the design and its reasoning, making it easier to comprehend the overall design.

There are overheads in creating an additional model to support design reasoning. The question is if the costs justify the benefits. At this stage, we do not have empirical data to support the costs and benefits of this method. However, if the reasoning model is built during the design process, the time taken to create the reasoning models should be a lot less than reverse engineering because the designers already have the background of the system and the UML entities do not need to be reentered again. On the other hand, if design issues were not uncovered at the early stage, it would be a lot more costly to fix them.

9.5.4 Benefits of Design Reasoning

During the course of the architectural design, the software architects had created a traceability matrix between requirements and design components. The architects initially thought that such traceability matrix would allow them to thoroughly analyze and design the system, it turned out that it was only partially useful. In summary, the researchers are able to pinpoint architectural design reasoning gaps in the design. Through analyzing the gaps, we have noticed that ad hoc design decision making does not provide a systematic approach to architectural design. As a result, conflicts in requirements and design have occurred, certain requirements are ambiguous, and decisions have not been well-thought out. The results of this study have indicated that design reasoning can help to achieve the goals described in Table 9.1, a summary is shown in Table 9.4.

Table 9.4 How design reasoning process serve architectural design

Architectural activities	How design reasoning benefits architectural design
<i>Deliberating design</i>	<i>Associating</i> interrelated design concerns to identify a design issue, or a design topic, is an essential step in architectural design. It formulates what decision has to be made, from that architects need to <i>identify</i> the design options that can address the design issues.
<i>Justifying design decisions</i>	Having <i>identified</i> the design options, architects should justify why a certain option is chosen and how it satisfies the design concerns, using argumentation or trade-offs analysis method. If a design concern cannot be fully satisfied, then <i>backtracking</i> to change previously made decisions allow the architects to iteratively improve the design.
<i>Structured design process</i>	The design reasoning process supports a structured approach to software architecture design. It is an improvement that allows architects to synthesize a design by exploring and associating design concerns systematically.
<i>Design validation</i>	Design reasoning has allowed systematic analysis of the architectural design by the researchers, the method enables architects and reviewers to find hidden design issues, and serves to validate the architecture design.
<i>Communication and knowledge transfer</i>	The architects have noted that the AREL UML representation can be used to communicate and discuss design reasoning process and trade-offs with the stakeholders. The decision diagrams can also serve as documented agreements between stakeholders.
<i>Support architectural maintenance</i>	The architects have noted that capturing the design rationale can help new staff to understand the system and quickly becoming productive.

9.5.5 Limitations in the Case Study

A qualitative research method is used in this empirical case study. The researchers reverse engineered the design reasoning models of the system using its requirement and design specifications. The design reasoning model was used to assess if a design reasoning process could improve the design process. A direct application of the design reasoning method to the system would have been the preferred approach for testing this method. If another design team that does not use the method carries out the same design in parallel, the comparison between the two would yield directly comparable results. However for sizable real-life system development, this is almost impossible because of funding issues. As such, we have opted to use the reverse engineering method as a mean to obtain design data for comparisons.

One could argue that the researchers can benefit from the hindsight. However, the designers have more experience in the application domain than the researchers. Without a similar background, the researchers' only method is the design reasoning approach. The researchers were not aware of the possible issues that might appear in

this type of applications. Therefore, the design issues that have been uncovered by the researchers to a large extent can be attributed to the design reasoning technique.

Although the design team has more experience in this domain, the researchers have similar years of general design experience, so it is possible that such experience may bias the findings. It is very difficult to distinguish to what extent design experience or design reasoning attribute to a high quality design. An empirical study has shown that design reasoning can help inexperienced designers to design better [318]. Using the results from both studies, we suggest that the need for experience and design reasoning is relative to the complexity of the architectural design. Although we cannot distinguish which one of the two factors (i.e. reasoning or experience) is more important, we think that highly complex problems would require design reasoning techniques to aid the thought process.

9.6 Summary

In this chapter, we have outlined a design reasoning method that comprises of five steps. This method is iterative in nature, and it is based on the causal relationships between *design concerns*, *design decisions* and *design outcomes*. Using the AREL model, we have applied this design method to a real-life system. In the empirical study, it has been found that design reasoning steps to *associate design concerns* and to *identify design options* are the two important reasoning steps in design.

The architects involved in this case study were interviewed at the end of the study. They have confirmed that by using design reasoning, it helps them to deliberate and negotiate design, determine trade-offs and review the architectural design. They have also noted that the graphical representation of AREL is simple, thereby making the design reasoning process intuitive and easy to use. From this observation, we suggest that such simplicity can overcome some of the issues faced by existing design rationale models where complex representations have hindered their implementation [79, 58, 206].

Acknowledgements This research has been partially sponsored by the Dutch Joint Academic and Commercial Quality Research and Development (Jacquard) program on Software Engineering Research via contract 638.001.406 GRIFFIN: a GRId For inFormatIoN about architectural knowledge.