

# Software Architecture Design Reasoning: A Case for Improved Methodology Support

Antony Tang, Jun Han, and Rajesh Vasa, *Swinburne University of Technology*

Capturing and recording the reasoning behind software architecture design can encourage architects to more carefully consider design decisions and better support future maintenance.

**S**oftware architects make a series of design decisions when developing a system's architecture. Although the design decisions affect the system's quality, architects often fail to capture the rationale behind the decisions. Instead, they focus on the design's outcome. The lack of commonly accepted methodologies or standards to guide systematic design reasoning frequently leaves architects to rely on their experience and personal preferences in making design decisions. Consequently, it's the experienced architects who are more likely to produce better designs, because

they know intuitively (but not always predictably) what key issues to consider.<sup>1</sup>

In general, design reasoning and decision making are still tacit processes, much of which architects learn on the job. Exploring what constitutes a good decision should lead to a better-understood and improved design process. Design reasoning has two key elements: explaining a decision's context and outcome and justifying a decision's reasoning. Identifying and recording this information encourages architects to carefully consider their design and document evidence to support future maintenance.

Previous research has shown that design rationale documentation helps designers understand and reason about their design.<sup>2,3</sup> Using AREL (Architecture Rationale and Elements Linkage), a UML-based model, we explore architectural design reasoning and how its elements relate to requirements and design models. We apply AREL in a case study using a real-life electronic payment system and compare design-reasoning support with and without AREL.

## Software Architecture Design Reasoning

Software architecture is the "fundamental organization"<sup>4</sup> or "structures"<sup>5</sup> of a system's components. Architectural design is an early activity in the development cycle, and it's often difficult to verify before system implementation. Consequently, rectifying any mistakes or omissions in an architecture's design can be costly.

In software architecture design, architects apply cognitive reasoning, although they might not think about it consciously. Understanding the reasoning process can help tremendously in delivering good design. For example, designers' judgment can be inadvertently biased owing to the availability of information and personal preferences or experiences, a common situation that can adversely affect system quality.

On one hand, architects have built many systems successfully without explicitly employing design-reasoning methods. These successful projects often rely on people with experience and good judgment.

**In 1958,  
Stephen Toulmin  
suggested  
that we can use  
argumentation  
to induce  
conclusions  
from contextual  
data.**

Some IT professionals seem to have an uncanny way of foreseeing problems, formulating solutions, and making just the right decisions consistently. On the other hand, many practitioners design poorly, overengineering a solution, underestimating the effort, missing key requirements, selecting the wrong technologies, or delivering poor-quality design. The challenge is how to systematically improve designers' reasoning abilities to consistently deliver a satisfactory design and to improve architecture design's quality-assurance process.

A well-designed architecture should be justifiable through sound, logical design rationale. Design reasoning ought to consider all relevant architecture requirements, address the design issues, and consider trade-offs among design options before decision making. The purposes for having such a rationale-based architectural design approach in the development life cycle fall into two categories: support for software architecture design and support for maintenance activities.

The following benefits support software architecture design:

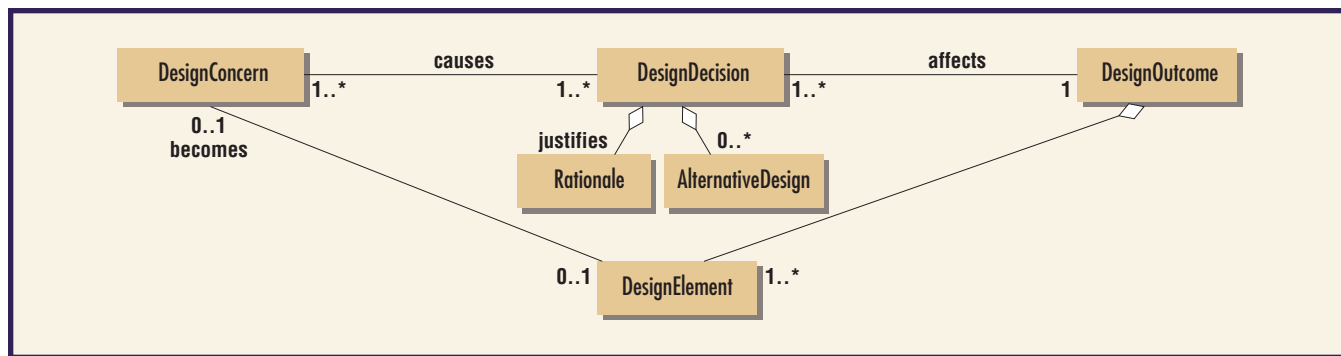
- *Deliberating and negotiating design.* Design rationale lets designers systematically clarify the issues and possible solutions and evaluate decisions against well-defined criteria. As such, it lets designers and stakeholders deliberate and negotiate a solution.
- *Justifying design decisions.* Design rationale can explicate tacit assumptions, clarify dependencies and constraints, and help justify selections.
- *Applying trade-off analysis.* A design decision often involves resolving conflicting requirements that the development team can't fully satisfy simultaneously. With an architectural trade-off analysis method such as ATAM,<sup>5</sup> the prioritized requirements and utility tree form the compromised decision's reasoning.
- *Providing a structured design process.* Design rationale supports a structured, accountable design process. It provides an understanding of a project's context, stakeholders, technologies, and situations.
- *Enabling design verification and review.* Design rationale explains design decisions and provides the information needed for independent architecture design validation and review.
- *Facilitating communication and knowledge transfer.* Design rationale can help business analysts evaluate conflicting requirements and new designers learn the architecture design.

The following benefits support maintenance activities:

- *Retaining knowledge.* If system maintainers aren't the same people who originally developed the system and the design rationale is unavailable, maintainers would have to second-guess the intangible rationale.
- *Understanding previous design alternatives.* Design alternatives can help maintainers understand why the architects deemed certain design options unviable. They can also enable maintainers to consider alternatives that once were unviable but are now viable.
- *Understanding design dependency.* Design decisions can be interdependent and cut across numerous issues. Changing a decision might have ripple effects on other system parts. Recording design rationale and its interdependencies helps avoid overlooking related issues.
- *Improving maintenance productivity.* One experiment showed that a group of designers equipped with the design rationale can work faster and identify more changes required in a system maintenance exercise than a control group without the design rationale.<sup>2</sup>
- *Predicting change impact.* Design rationale could help maintainers predict which system part is subject to consequential change.<sup>6</sup>
- *Providing traceability.* With an explanation of the process, maintainers would be able to trace how design artifacts satisfy system requirements.<sup>7</sup>

Software architecture and development standards such as IEEE 1471-2000 and ISO-15288 have suggested that design rationale and decision strategy are important in software development. However, they have offered little guidance on how to achieve a reasonable design-decision process. Similarly, architecture frameworks, such as the Open Group Architecture Framework, Federal Enterprise Architecture Framework, and the Zachman Framework, have recommended that architects understand the reasons behind an architectural design. Yet, such frameworks and standards have few suggestions on what design rationale to capture and how to capture it.

In 1958, Stephen Toulmin suggested that we can use argumentation to induce conclusions from contextual data. This approach explicitly represents the design deliberation process to relate design goals to design results. Methods using this approach include Issue-Based Information Systems; the Decision Representation Language; and Questions, Options, and



**Figure 1. The conceptual model of design reasoning in Architecture Rationale and Elements Linkage. AREL captures the design rationale and the design model.**

Criteria.<sup>8</sup> Unfortunately, these methods have been unsuccessful in practice because they have difficulty capturing and communicating design rationale.

Another approach, using templates<sup>9,10</sup> to help capture design reasoning, is receiving industry attention as practitioners recognize the importance of recording design reasoning. UML case tools, such as Rational Rose and Enterprise Architect, provide easy-to-use facilities to capture design rationale. (Design-rationale support in Enterprise Architect uses AREL plug-in tools.) Although template-based methods can capture design rationale, they provide limited support to relate design decision contexts.

We developed AREL to make design reasoning easy for software development organizations to adopt without losing argumentation-based methods' design-reasoning capabilities. AREL is a hybrid approach that incorporates design rationale templates and design-reasoning relationships based on previous research.<sup>8-10</sup> We designed it to capture useful design-reasoning information with minimal documentation overhead.

## Supporting Architecture Design Reasoning with AREL

What exactly is design reasoning? Is it a reason for having a system, or is it justification for how a system is designed? First, let's consider a simple reasoning model that comprises three elements. *Inputs* are the requirements and goals that a system must meet. *Decisions* are those that architects make in designing the system. *Outputs* are the design's results. Without inputs, we would miss out on the contextual information that tells us why we need the design. Without design decisions, we might not understand the justifications or reasons for choosing design outputs. Moreover, we need to know the causal relationships between the inputs, decisions, and outputs to understand how and why a design is constructed. So, design reasoning is a description of the design context, justification, and outcome and their causal relationships.

IEEE-1471-2000 calls the inputs *architectural design concerns* and defines them as those interests that pertain to the system's development, its operation, or any other aspects important to one or more stakeholders. Examples include a system's security and performance requirements. The standard also specifies that architectural rationale should provide evidence for considering alternative architectural concepts and the rationale for the choices made. The standard doesn't provide much detail to help implement a design-reasoning model. In a discussion session on updating the IEEE-1471 standard, participants suggested refinements to overcome this issue.<sup>11</sup>

AREL captures both the design rationale and the design model using Enterprise Architect. The design model includes architecturally significant requirements and the architecture design. AREL associates design decisions to a design model using the following concepts:

1. Design concerns raise a design decision.
2. A design decision captures design issues and design rationale.
3. Design outcomes are the results of a design decision.

Figure 1 shows AREL's conceptual model. Design concerns such as requirements cause the need for design decisions. When an architect makes a design decision, the design rationale justifies the decision and contains any discarded alternative designs. These alternative designs can help reviewers and maintainers examine the considered options. Design decisions affect the design outcome, and the design outcome contains the chosen design elements.

A chosen design element is a UML entity to model the design, such as class, component, or database table. Each design element can influence other system parts because of its behavior or constraints. These design elements will in turn become design concerns that influence other parts of a

**After tracing the AREL diagrams, the architects could explain the asynchronous messaging's design rationale.**

design. For instance, choosing Ajax as a solution to construct Web pages leads to the issue of having to handle the Web browser's "back" button. Ajax as a solution, therefore, becomes a new design concern. Such design consequences could trigger a chain of interdependent designs and design decisions.

### Design Concerns

Capturing a design decision's causes is vital to comprehending a design's reasons. The inputs that cause or motivate a decision are design concerns. Design concerns represent a design's context and can be anything that influences the design decision. Functional and nonfunctional requirements are examples of design concerns. However, many design concerns are often omitted by architects, although they significantly impact the system architecture. For instance, in the case of usability, many designers might implicitly use relevant principles without articulating how they're related to the design. These principles are relevant in design reasoning because ignoring them would lead to questionable design decisions.

The different types of design concerns influence decisions differently. *Requirements* drive and motivate design creation. *Purposes and goals* provide a context to guide the design. *Environmental factors* constrain the available choices of an architecture design. More specifically, design concerns include these:

- *Purposes and goals*—a system's business goals.
- *Functional requirements*—a system's functional goals.
- *Nonfunctional requirements*—quality attributes that a system must fulfill, such as performance and usability.
- *Business environment*—organization and business environmental factors that influence architecture design, such as long-term or strategic organization goals.
- *Information system (IS) environment*—environmental factors that influence the system's construction and implementation, such as budget, schedule, and expertise.
- *Technology (IT) environment*—technological factors that influence the architecture, such as current organizational technologies and policies.
- *Design*—a chosen design (outcome) that influences the rest of the architecture. For example, the selection of an operating system constrains development tool choice.

For instance, when an architect designs a business-to-business Web site, environmental design concerns could include the business environment (outsourcing a system has implications on the maintainability and support requirements), IS environment (time to market is three months), and IT environment (organization standards to use ASP, Net and Oracle). These concerns exert certain constraints on the design decisions.

### Design Decisions

Making design decisions through systematic reasoning requires explicit justification using design rationale. Documenting design rationale for the entire system can be costly and probably unnecessary, so we focus on architectural issues that are often complex and intertwining and require much investigation.

Design rationale is the reasoning 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. One issue might simply be designing to satisfy some design concerns. For instance, what data must I 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? More than one possible solution might exist. 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.

To capture such justifications, AREL uses qualitative and quantitative design rationale. Qualitative design rationale considers these factors:

- *Design issue*—the issue to be dealt with in a decision.
- *Design assumptions*—the decision makers' assumptions.
- *Design constraints*—the technical or contextual constraints on a decision.
- *Strengths and weaknesses*—a design option's strengths and weaknesses.
- *Trade-offs*—a balanced analysis of what's an appropriate option after prioritizing and weighing design options.
- *Risks and nonrisks*—considerations about a design option's uncertainties or certainties.

It supports design reasoning by way of arguments. Architects can document a decision's justifications

by arguing design options' relative advantages and disadvantages or using a trade-off analysis method such as ATAM.<sup>5</sup>

Quantitative design rationale considers these factors:

- **Cost**—the relative cost in areas such as development efforts, platform support, maintenance cost, and other intangible costs such as potential legal liabilities.
- **Benefit**—how well a design option might satisfy the requirements and quality attributes.
- **Implementation risk**—the risk that a development team might not implement the design successfully for reasons such as the lack of capability or experience.
- **Outcome certainty risk**—the risk that a design might not satisfy the requirements because they're technically unachievable or not well defined.

It records a design option's relative costs, benefits, and risks as ordinal numbers between 1 (lowest) and 10 (highest). Capturing quantitative rationale allows a quantifiable comparison between alternative design options and lets architects highlight risky or costly decisions that must be investigated further. For example, if the implementation risk is high, architects might continue to decompose the architecture through a series of decisions until the risk becomes manageable.

## Design Outcome

Design decisions' results are chosen designs. A chosen design outcome can become a design concern because it can create a new design problem, as in the earlier Ajax example. So, a design outcome is associated with the next decision if it influences that decision.

In practice, architectural design can take various forms, such as PowerPoint drawings, UML diagrams, or formal specifications. The forms vary because an architecture design describes a variety of issues and because the definition of what should be in an architectural design is subject to interpretation. At a high level, it describes technology platform choices, system functionality layering, software component packaging, runtime processes, and so on. At a low level, it describes the structures and relationships of components, classes, interfaces, data stores, and so on. AREL uses basic UML entities such as package, object, class, and component to represent the architectural design outcomes.

## Applying AREL

To demonstrate software architecture design reasoning using AREL, we applied it to a real-life system and conducted experiments about its use.

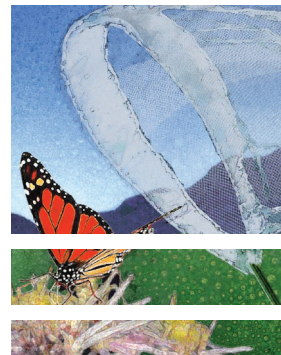
### Modeling with AREL

We selected Guangzhou Electronic Banking Settlement Centre's electronic-fund-transfer (EFT) system, because it's a large project with significant architectural challenges, including security, performance, correctness, and efficiency. (The Guangzhou Electronic Banking Settlement Centre is a division of the central bank in China and a key service that member banks in southern China need.)

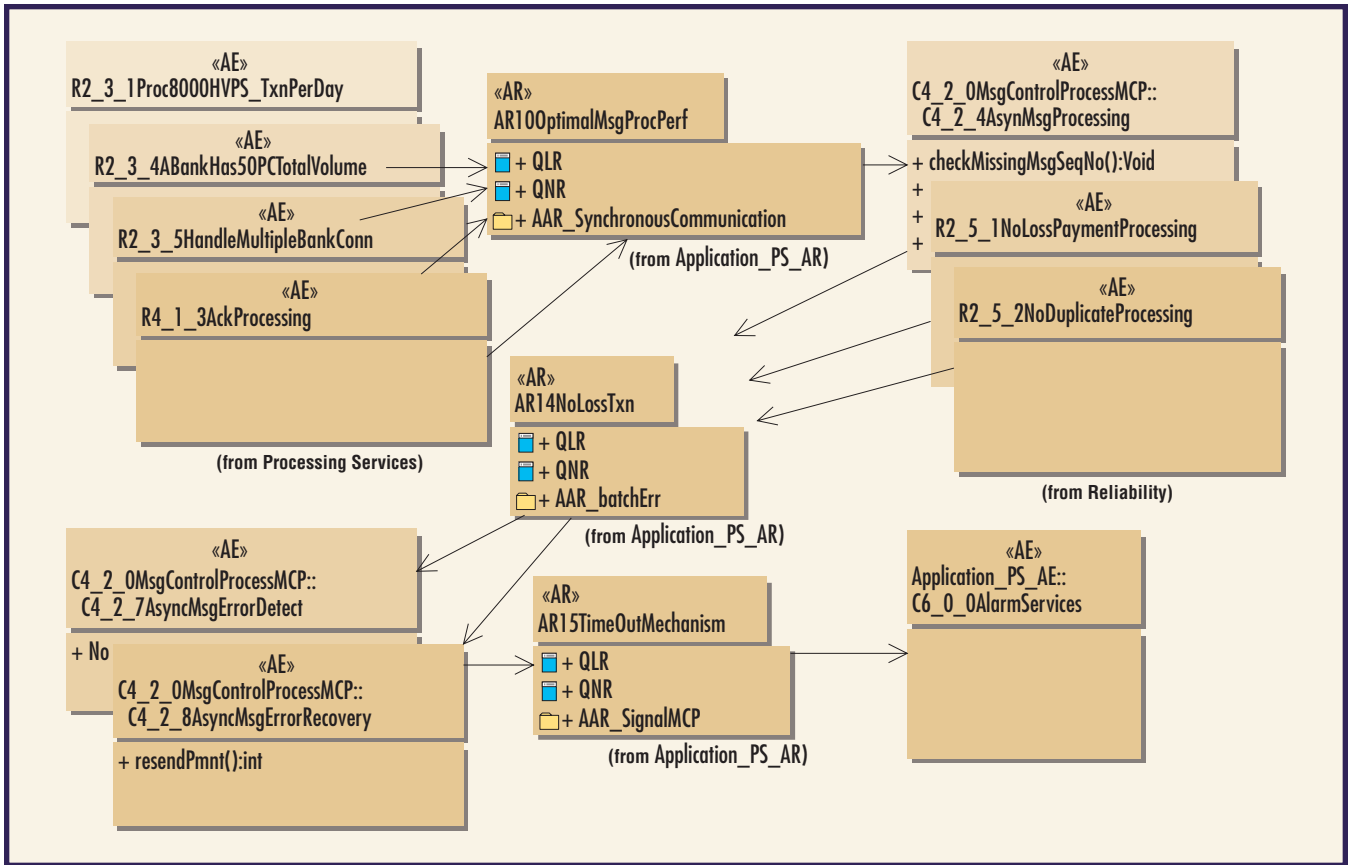
A design decision in AREL contains the architecture rationale (<<AR>> in Figure 2 on the next page), providing the decision's justification. The decision is related to its context—that is, design concerns are the inputs, and design outcomes are the results. We model design concerns and design outcomes as UML entities (<<AE>>). In this example, we retrospectively modeled the system's payment message-exchange mechanism. As Figure 2 shows, three related decisions addressed the design issues.

The first design issue was to find a message exchange protocol that would optimize messaging performance. The options were to adopt either a synchronous or asynchronous message exchange. Performance (R2\_3\_1) and acknowledgments handling (R4\_1\_3) were design concerns we had to consider. We captured the design rationale in the qualitative rationale (QLR) and quantitative rationale (QNR) in AR10, which the user can view by clicking on the AR10 icon. An alternative design package (prefixed by AAR) captured the discarded design option and its design rationale. It was decided to use an asynchronous protocol because it provided higher concurrency and throughput. This approach's weakness, however, was its design complexity. We captured the trade-off between designing an efficient or simple system, using the QNR template. QNR captures architects' estimates of the design options' costs, benefits, and risks. As a result of this decision, the architects created a module to process asynchronous message exchange (C4\_2\_4).

At this point, the architectural design was incomplete because relevant design was still missing, and the implementation risk of AR10 was high (captured in QNR). So, we continued with the next design issue: guaranteeing no loss of transactions during message exchange. Because payment messages arrive asynchronously, if an acknowledgment is missing, the system must decide whether







**Figure 2. EFT message-processing architecture (partial) in AREL. The architecture rationale (<<AR>>) justifies the decision made for the design concerns and outcomes (<<AE>>).**

it is really missing or it is just arriving late. Asynchronous processing (C4\_2\_4) was the outcome of AR10 and now is one of the concerns for decision AR14. The AR14 decision created two modules to cater for missing message detection (C4\_2\_7) and message recovery (C4\_2\_8). Because error detection (C4\_2\_7) relies on a message's time-out, the third and subsequent design issue was to select a mechanism to implement time-out. AREL captured this decision in AR15 and consequently an alarm server (C6\_0\_0) was created.

This example illustrates three of AREL's main characteristics. First, it links a decision's causes and effects in a general form (design concerns → design decision → design outcomes). Second, a decision contains the design rationale and discarded design options. Finally, decisions are interrelated if a causal relationship exists between a decision's design outcome and its subsequent decision.

### Architect Feedback

To test whether AREL can help architects understand a design's rationale, we asked nine architects and designers with electronic-payment-systems experience to participate in an experiment. Some of them were involved in the original EFT system design. They had an average of 16.6 years' experience


in the IT industry, with an average of 7.7 years' experience in designing payment systems. We asked them to analyze the EFT system using its original design specifications and using AREL. We then compared the results to see whether the AREL models provided additional information to help the architects understand the software architecture design.

We presented four areas of the EFT system design to the architects, including the message-processing design (Figure 2). First, all the architects read the original design specifications. We then asked them to explain why the EFT system architects chose the particular design. For asynchronous-messaging design, after searching and analyzing the specifications, they couldn't state reasons with any certainty. However, they all suggested the design was related to system performance.

Next, we introduced the architects to the AREL diagrams and asked them to explain the design reasoning. After tracing the AREL diagrams, the architects could explain the asynchronous messaging's design rationale. They traced the design concerns to the design decisions and outcomes, and they explored the design rationale the decisions contained.

This new information led many architects to comment on the design's complexity. Although asynchronous messaging reduces message blocking, it created additional requirements. The choice of the initial decision seems sound, but the chain of decisions eventually created a complex solution. Some architects suggested a multithreaded solution as an alternative. They discovered this alternative during the case study because they were able to trace interdependent design decisions diagrammatically and backtrack to the decisions where they could consider alternatives. Those architects who are still involved with maintaining the system said this new understanding enables them to reconsider their design. The result demonstrated that it's possible to better understand and improve an architectural design if design rationale is captured and the relationships between interrelated designs are traceable.

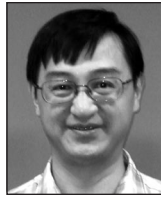
We found in a survey that design rationale wasn't systematically documented despite established corporate development standards.<sup>12</sup> Without explicit documentation of the design reasoning, most architects were uncertain about what design concerns might have influenced the design and what issues and justifications the original architects considered. In addition, design rationale is useful mostly for intricate designs. Among the four design areas in this experiment, design specifications alone were sufficient to explain two of them; the other two more complex designs required AREL for explanations.

**O**ur experiment's findings are a positive step toward overcoming the communication difficulty in some argumentation-based methods. We based our experiment on a single system and therefore can't generalize its benefits to all types of architectural design. However, it does demonstrate the approach's potential application to the design of large, complex systems. To further explore this issue, we're continuing our research with industrial projects. 

## Acknowledgments

We thank Z.T. Duan and his team at the Guangzhou Electronic Banking Settlement Center for their support in our research. We also thank the practitioners who participated in our study. Finally, we thank Sparx Systems for providing Enterprise Architect to support our work.

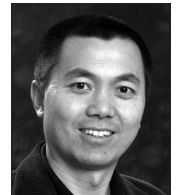
## About the Authors



**Jun Han** is a professor of software engineering at Swinburne University of Technology. His research interests include software architecture design, adaptive software systems, software security engineering, software performance engineering, system integration and interoperability, and services engineering. Han received his PhD in computer science from the University of Queensland. He's a member of the IEEE Computer Society and the ACM. Contact him at [jhan@swin.edu.au](mailto:jhan@swin.edu.au).



**Antony Tang** is a senior lecturer at Swinburne University of Technology. His research interests include software architecture design reasoning and software development. Tang received his PhD in IT from Swinburne University of Technology. He's a member of the IEEE Computer Society and ACM. Contact him at [atang@swin.edu.au](mailto:atang@swin.edu.au).



**Rajesh Vasa** is the International Program Manager at Swinburne University of Technology. His research interests include software evolution and change visualization. Vasa received his bachelor of applied science in computer science and software engineering from Swinburne University of Technology. Contact him at [rvasa@swin.edu.au](mailto:rvasa@swin.edu.au).

## References

1. A. Tang et al., "Design Reasoning Improves Software Design Quality," *Proc. 4th Int'l Conf. on the Quality of Software Architectures*, LNCS 5281, 2008, pp. 28–42.
2. L. Bratthall, E. Johansson, and B. Regnell, "Is a Design Rationale Vital When Predicting Change Impact? A Controlled Experiment on Software Architecture Evolution," *Proc. 2nd Int'l Conf. Product Focused Software Process Improvement*, LNCS 1840, Springer, 2000, pp. 126–139.
3. L. Karsenty, "An Empirical Evaluation of Design Rationale Documents," *Proc. SIGCHI Conf. Human Factors in Computing Systems: Common Ground*, ACM Press, 1996, pp. 150–156.
4. *IEEE Std. 1471-2000, IEEE Recommended Practice for Architecture Description of Software-Intensive Systems*, IEEE, 2000.
5. L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed., Addison-Wesley, 2003.
6. A. Tang et al., "Using Bayesian Belief Networks for Change Impact Analysis in Architecture Design," *J. Systems and Software*, vol. 80, no. 1, 2007, pp. 127–148.
7. B. Ramesh and M. Jarke, "Towards Reference Models for Requirements Traceability," *IEEE Trans. Software Eng.*, vol. 27, no. 1, 2001, pp. 58–93.
8. T. Moran and J. Carroll, *Design Rationale: Concepts, Techniques, and Use*, Lawrence Erlbaum, 1996.
9. P. Clements et al., *Documenting Software Architectures: Views and Beyond*, Addison-Wesley, 2002.
10. J. Tyree and A. Akerman, "Architecture Decisions: Demystifying Architecture," *IEEE Software*, vol. 22, no. 2, 2005, pp. 19–27.
11. P. Avgeriou et al., "Architectural Knowledge and Rationale—Issues, Trends, Challenges," *ACM SIGSOFT Software Eng. Notes*, vol. 32, no. 4, 2007, pp. 41–46.
12. A. Tang et al., "A Survey of Architecture Design Rationale," *J. Systems and Software*, vol. 79, no. 12, 2006, pp. 1792–1804.

For more information on this or any other computing topic, please visit our Digital Library at [www.computer.org/csdl](http://www.computer.org/csdl).