Architecture Rationalization: A Methodology for Architecture Verifiability, Traceability and Completeness

Technical Report: SUTIT-TR2004.05
CeCSES Centre Report: SUT.CeCSES-TR005

Antony Tang, Swinburne University of Technology
Jun Han, Swinburne University of Technology
1st September 2004
# Table of Contents

Abstract 1

1. Introduction 1

2. Background and Related Work 2
   2.1. Viewpoints 3
   2.2. Verifiability 3
   2.3. Traceability 4
   2.4. Differentiating Architecture and Detailed Design 4

3. Architecture Rationale 5
   3.1. Elements of Architecture Rationale 6
   3.2. Qualitative Rationale 6
   3.3. Quantitative Rationale 7
   3.4. Rationale Evaluation 8

4. Architecture Rationalization Method 9
   4.1. Requirements Refinement 9
   4.2. Architecture Decomposition 10
   4.3. Architecture or Detailed Design 11

5. A Retrospective Case Study 12

6. Conclusion 17

7. References 19
Abstract

Architecture modeling is practiced extensively in the software industry but there is little attention paid to the traceability, verifiability and completeness of architecture designs and the rationale behind those designs. Deficiencies in any of these three areas in an architecture model can be costly and risky to projects. We propose the Architecture Rationalization Method (ARM) to overcome these issues. ARM makes use of both qualitative and quantitative rationales for selecting architecture designs. Quantitative rationale uses a model based on costs, benefits and risks in the selection process. ARM provides a method to determine when an architecture model is complete in that the level of details represented by the architecture design is sufficient. We apply ARM to a real-life industry case retrospectively to demonstrate how ARM can overcome issues surrounding traceability and verifiability.

1. Introduction

It is common practice in the software industry to carry out architecture modeling as part of the software development life cycle (SDLC). The current trend in large-scale software development is to use architecture frameworks in designing the structure of a software system. Architecture design is a crucial step in the SDLC. Despite its importance, there are very little, if any, study and practical methods to verify architecture designs, to trace design rationales, or to ensure the architecture is complete.

Despite having software development processes, largely the software community designs software and system architectures based on experience and intuition without having to justify why the software is architected in a certain way as long as systems meets requirements. The lack of guidance on design rationalisation is also evident in architecture frameworks and software standards. Artifacts from architectural activities always describe what is to be built and how to build it. The documentation of the design thought process is often omitted. In this case study, we examine the prototype of a check clearing system built four years ago. The first author developed the prototype and it was handed over to another team of designers. The lack of documented architecture rationale made it difficult for the team which took over the project to understand the design and implement the full product, even though the architecture and design specifications and the prototype programs were available. This difficulty has led to an unfortunate decision to abandon the prototype and restart anew. This scenario is quite common in the software industry. The key issues underlying such cases are that (1) architecture rationales were not recorded, and (2) consequently the reasons for the architecture decisions could not be traced or verified. Vital information gathered during the design process such as assumptions, considerations and tradeoffs, which is essential to the understanding of the architecture, was missing.

Given the issues underlying architecture designs, we analyse the key elements in architecture modeling to resolve them. The act of architecture is to produce architecture models, it deals with...
structural issues of the system and its designs are architectural designs. A collection of architectural designs forms the basis of an architecture model. Architecture designs are distinct from detailed software or system designs in that they define the structure of the system and they do not consider issues that elaborate detailed designs. In this paper, when we refer to design, we mean architectural design and when we refer to detailed design, we mean detailed software or system design. The architecture completeness issue described in this paper will make a clear distinction between the two activities.

Our motivation is to represent architecture rationale of the system and incorporate it into the architecture process to address the issues of verifiability, traceability and completeness of architecture designs. It is important to note that architecture rationale is the fundamental enabler to address these issues. Many systems that exist now have some form of traceability between designs and requirements through cross referencing, but the lack of architecture rationale to support the trace makes it difficult to understand the relationship between them, why particular design was chosen and what alternatives had been considered. Furthermore, there has to be some tangible criteria to determine when an architecture model is complete where no additional work is required at this stage. We apply our newly developed method retrospectively to the check truncation system to show what additional information would help designers to understand the original architecture design and verify that it is sound.

Architecture rationale has been identified by researchers and standards bodies as crucial in the architecture design process [1, 2], but none of the architecture frameworks surveyed [3] prescribe rationale as part of architecture deliverables and the nature of the architecture rationale is unclear. Another important aspect of architecture rationale is verifiability since correct design decisions made at the early stage of development have the greatest impact on the system [2]. Our contributions in this paper are the following:

- Introduce the Architecture Rationalization Method (ARM) to enable architects to systematically rationalize and record architecture decisions. ARM can be used in conjunction with generic architecture frameworks
- Make use of the Architecture Rationale (AR) for recording qualitative and quantitative rationales to facilitate architecture verifiability
- Provide a way to trace requirements mappings, requirements to design mappings and design mappings through AR for architecture traceability
- Propose a risk-based method to define architecture completeness so that architects can determine the scope of the architecture design and as such separate its activities from detailed design.

Section 2 describes the background of this work, current issues and related work in this area. Section 3 describes the elements of architecture rationale. Section 4 describes the ARM methodology and its application. We present the partially redesigned check truncation system using ARM as a case study in section 5 and we make some concluding remarks in section 6.

2. Background and Related Work
The challenges in large-scale systems architecture are to make balanced and correct design decisions in a complex and often conflicting environment. Conflicts arise from large numbers of
competing functional requirements, users with different objectives, diverse processes and interfaces, competing non-functional requirements such as performance, distribution, reliability and so on. Viewpoints used in Architecture Frameworks help to model complex systems. They take into account system requirements, information, computation and implementation modeling.

2.1. Viewpoints

An architecture model of a system consists of a number of views and together they represent the architecture design of the system. Different architecture frameworks (AF) such as RM-ODP [4], FEAF [5], TOGAF [6] and DoDAF [7] prescribe slightly different viewpoints for architecture modeling. The following are a generalization of the viewpoints:

- **Business / Enterprise View** – describes scope, policies, activities, business requirements and flows of business information between people and business processes. Both functional requirements and non-functional requirements can be documented in this view.
- **Information / Data View** – describes semantics of information in a system such as logical data model
- **Computation / Application View** – describes the decomposition of the system into components and their interfaces
- **Engineering / Technology View** – describes the mechanisms and choices of technology to implement the system.

Different views represent architectural designs from different perspectives. Views are results of design considerations, tradeoffs and decision making.

2.2. Verifiability

The architecture process involves a series of decisions making based on design choices, tradeoffs and compromises. Architecture verification checks that the architecture model is complete and the rationale of design decisions is sound and the design can satisfy system requirements. Current practice of verification in the industry is through peer review of architecture design specification in the early phase of the project. The rationale of design decisions is seldom documented. However, architecture decisions should be verifiable during or after development, with or without the presence of the original architects. Architecture rationale is the vital information in the verification process. The need to verify architecture is recommended by IEEE standards [8]. There are a number of research work related to architecture design rationale, architecture evaluation and the application of economic considerations [9, 10]. These methods use architects and stakeholders to provide the information in the verification process. In this paper, we say that an architecture design is verifiable if:

- There is a documented reason(s) of why the design can satisfy the requirement(s) and
- A design can be traced to other designs and / or requirements

Although there are suggestions as to what information should be captured in architecture rationale, there are few considerations as to how architecture rationale should be represented in architecture models and what should be the minimum requisite for verification. AR proposed here contains specific information to use in the verification of architecture models. The act of verification is still a
manual one although tools can be used to automatically extract design rationales that are associated with requirements and designs.

2.3. Traceability
An architecture model, as represented by views, is usually documented without cross referencing between system requirements and design elements. Architects would consider business requirements and system requirements in order to select an appropriate design from different choices by using reasoning, experience and intuition. However, the reasons are often not documented and even if some software development processes provide traceability between requirements documents and design documents, the relationship between requirements and design elements across views cannot be explained. The lack of traceability and design reasoning inhibit the ability to verify architecture design.

Our experience in this case shows that the lack of traceability has little impact on the initial implementation of a system, but it inhibits another team of designers to properly understand why the design is done in a given manner. Also noted in [11], traceability “provides critical support for system development and evolution.”. The need to ensure that requirements are allocated, or traced, to software and hardware items is recommended by the IEEE standards [8, 12].

2.4. Differentiating Architecture and Detailed Design
Perry and Wolf [1] state that architecture deals with load bearing issues as against decoration issues. This view is confirmed by the IEEE standard that architecture is about “fundamental organization of a system” [2]. The obvious questions are [3, 11]:

- Based on what criteria should an architect consider architecture modeling complete? i.e. what is the scope of architecture?
- To what level of design details would architecture modeling be considered complete and from where do detailed design activities start? i.e. what is the depth of architecture?
- Does every part of the system require the same amount of architectural design detail?

There has been some suggestions in this area [13] but our survey shows that there is no distinction between architecture and detailed design made by any of the architecture frameworks [3]. The lack of understanding of this area implies that (a) architecture design activities could go beyond what is necessary into detailed design activities, i.e. over architecture; (b) the architecture design could be incomplete thereby posting risks to the quality of design; (c) the completeness and quality of the architecture design cannot be verified at an early stage when any changes have the lowest impact on development; (d) roles and responsibilities of the architects and the software designers can be ambiguous in a project team. Based on these issues, we define architecture completeness so that we can clearly identify when architecture design activities are complete.
3. Architecture Rationale

Architecture Rationale (AR) is an artifact defined to record the reasons of requirements enhancements and design rationale. It has two primary characteristics to enable architecture verification and tracing. *Reasoning* represents the rationale behind an architecture choice to satisfy particular requirement(s). *Referencing* provides cross references, or mappings, between elements in an architecture model through an AR.

![Figure 1. Architecture Rationale Model](image)

The AR model is an UML representation depicted in Figure 1. Both instances of Architecture Rationale (AR) and Alternative Architecture Rationale (AAR) have a generic stereotype of <<AR>>. An instance of AR contains *reasons* of a particular design decision at a decision point and it *references* two or more elements contained in the views of an architecture model. For instance, a requirement in the Business View may be referenced, or mapped, to a design element in the Computation View through an AR. This is represented by associations using stereotype <<trace>> between an AR and at least two entities in the view model.

AR consists of qualitative and quantitative reasoning. Qualitative Rationale (QuR) describes in textual format the qualitative information about a design choice. Quantitative Rationale (QaR) provides a quantifiable justification based on costs, benefits and risks of a particular design. Additional scenarios could be associated with an AR to show constraints or special cases of a design. All these three entities are an aggregate part of AR.

AAR has all the properties of AR but this design has been rejected. The rejected design model of AAR is contained in Alternate Model. AAR is associated with the accepted AR. AAR is useful in a number of ways. It helps the architect to consider alternative design models; it documents rejected designs and the rationale for rejection; and it allows alternatives to be reconsidered in future system evolution. AAR also provides important information for architecture verification.
3.1. Elements of Architecture Rationale

AR is an artifact that documents the rationale of a design choice. Elements of AR are an aggregation of the following:

- An identifier to uniquely represent the AR
- At least two or more references to relate or map elements contained in the architecture view model. The mapped elements may be from the same view or from different views
- Any element in a view which is mapped to AR must be identifiable despite the level of details
- Qualitative rationale (see below)
- Quantitative rationale (see below)
- Scenarios to depict constraints or special cases
- AR is associated with zero or more AAR to show alternate designs
- AAR may contain design in Alternate Model for future reference

There should be as many AR as decision points in an architecture model to complete the model.

3.2. Qualitative Rationale

Choosing a design involves evaluation of design alternatives, model feasibility, benefits of design and compromises to be made. The decision making process is as important as the decision being made because it provides vital information for verification and subsequent change management. AR and AAR make use of qualitative rationales suggested in [9, 14] and we enhance and classify them into the following categories:

- Design constraints. They might be of a project nature such as budget, schedule, resource; or of requirements nature such as competing requirements; or of technical nature such as performance or capacity metrics
- Design assumptions such as expected system usage pattern or expected throughput
- Strengths and weaknesses of a design
- Tradeoffs that are compromises made between competing requirements or designs
- Risks and non-risks that document the known uncertainties or certainties of a design. Risk elements are quantified in QaR for comparisons
- Scenarios that record design limitations or exclusions where compromises have been made. This is separate from the design model because they are not adopted in the final system
- Assessments that are reasons or justifications behind the selection or exclusion of a design. This attribute is mandatory because architects should provide a balanced assessment after considering all factors in QuR and QaR.

Architects could use some or all of QuR categories depending on the context of the architecture design. For instance, AR may map a business requirement to a computational component, and then map to a non-functional requirement, the constraints and assumptions of the implementation of the non-functional requirement would be documented in the AR together with the assessment of the decision.
3.3. Quantitative Rationale

For most architecture designs, the decision making process is based on the experience and intuition of architects and the basis of decisions cannot be subsequently measured. Quantitative rationale enables systematic estimates of the likely impact of individual decisions at each decision point. The likely impact of a decision is represented by the architect’s estimate of Expected Return (ER) on the part of architecture model under consideration. The QaR approach is based on three elements – cost, benefit and risk.

The architecture costs and benefits are represented by two indices scaled from one to ten. The reasons for using an index instead of money value are because (a) some of the assessment cannot be expressed in money terms for they may be intangible or difficult to estimate; (b) there may be multiple factors in which their benefits or costs values cannot be combined; (c) the comparison of AR to AAR can be made using the same scale and (d) it provides a uniform measure for reviews and verification.

Architecture Cost Index (ACI) is a weighted index that refers to the costs of implementing the decision or design. ACI is an index that takes into consideration a multitude of cost factors to provide a relative cost index between alternative rationales. Considerations for ACI weighing are the following:

- Development costs take into account the amount of development, level of development complexity, current skill set and training requirement
- Platform costs take into account the cost for platform support such as hardware and software
- Maintenance costs take into account routine operational maintenance and support, software maintenance, software modifiability and portability
- Potential costs such as security, legal and other implications that may arise from the design should also be considered

Actual cost information to support the decision making process should be gathered and analyzed if possible and documented in the constraints section within QuR for future references.

Architecture Benefits Index (ABI) is a weighted index to represent the relative benefits an architecture decision or design would deliver to satisfy the concerned requirements. If compromises are made between competing requirements, then the architect would make a judgment on the relative priority of requirements and the level of satisfaction the architectural design provides in meeting the requirements. Similar assessments would be made for alternative designs. CBAM provides a method to calculate the benefit score [10], this may be necessary for certain types of systems but the cost of applying this technique may be too high for some systems development. ABI provides an effective way to assess and estimate the benefits of architectural designs.

There are two types of risks represented in ARM and they are represented by ratios ranging between zero and one. Outcome Certainty Risk (OCR) identifies the impact of the risk or the uncertainty level of the architectural design to meet the desired outcome represented by the Architecture Benefits Index (ABI). Implementation Certainty Risk (ICR) identifies the risk or the uncertainty that there are no unexpected issues in the implementation of the architectural design. In other words, ICR represents the architect’s assessment of the uncertainty that issues may occur during the design, development or implementation phases, thus affecting the certainty of the Architecture Cost Index (ACI).
The QaR approach has four merits (a) it uses quantification as part of the architecture rationalization process; (b) quantification of the rationale is localized at each decision point; (c) $ER$ considers risks or uncertainties in architecture modeling; (d) reviews can be performed on an architecture model to measure soundness of each decision. The use and interpretation of these elements are different to some of the existing quantitative models [10, 15] in that QaR requires architects’ quantification or assessment of the three elements. However, we believe that experienced architects would provide consistent estimates because the basis of these estimates is a conscious decision of tradeoffs between requirements, designs choices and risks that are supported by qualitative facts.

3.4. Rationale Evaluation

The decision making process of ARM relies on rationalization at each decision point for all the architecture requirements and designs. The evaluation of alternative rationales is made at each decision point and it is based on evaluating both QuR and QaR. QaR is the representation of quantified costs, benefits and risks of QuR. Quantitative rationale evaluation uses Expected Return Ratio ($ER$) to measure the expected benefits or returns of a design choice at a particular decision point. Let Expected Benefit ($EB$) be

$$EB = (1 - OCR) \times ABI$$

Expected Benefit ($EB$) is the Architecture Benefit Index ($ABI$) discounted by the impact of the risk of not meeting the benefits which is represented by $OCR$. Let Expected Cost ($EC$) be

$$EC = (1 + ICR) \times ACI$$

Expected Cost ($EC$) is the Architecture Cost Index ($ACI$) amplified by the risk of design implementation which is represented by $ICR$. Expected Return ($ER$) can then be expressed as follows.

$$ER = \frac{EB}{EC} = \frac{(1 - OCR) \times ABI}{(1 + ICR) \times ACI}$$

$ER$ is the weighted expected return a design would deliver. The higher the $ER$ ratio, the better the expected return. Given two different architectural designs that deal with the same set of requirements, the design that has a higher $ER$ ratio is necessarily the better choice because the architect would have considered all relevant factors in QuR before quantifying them in QaR. $ER$ is an index to indicate relative returns of each design, therefore architects would assign the ratings in the same scale to differentiate the design choices at a decision point to justify the selection.

Evaluation of risks is an important and useful aspect of architecture modeling because it provides a means to discover and identify uncertainties [16]. As such, we argue that it should be done as an ongoing activity in the architecture construction process. Some research work suggests that software architecture risk is evaluated when a model is relatively complete [17], but ARM risk analysis is carried out during the entire decision making process to evaluate and select from alternative designs at each decision point. As we shall see in the next section, architecture designs often influence requirements in a feedback loop and that means two alternative architecture designs at a decision point may try to solve the same problem but each may have different implications on
the functional or non-functional requirements. This feedback may lead to refinement of requirements.

Evaluation of architecture choices requires comparing their ER ratios as well as comparing the qualitative rationales that support them. The examination of QuR when making architecture choices is especially useful when the ER ratios of two alternative models are close to each other.

4. Architecture Rationalization Method

The Architecture Rationalization Method (ARM) is a methodology to rationalize architecture design based on architecture frameworks. ARM uses a top down approach in architecture design employing two techniques that focus on rationalization:

- A requirement refinement technique
- An architecture decomposition technique

Outcomes of ARM are architecture rationale (AR) that map related requirements and design elements. ARM is used with architecture frameworks and will complement existing architecture practices. AR as the additional architecture artifact can be added to and used in existing architecture knowledge base.

4.1. Requirements Refinement

Architecture may sometimes involve in defining initial business or functional requirements but an architecture design process often leads to refinement of functional requirements (FR) due to conflict resolutions or requirements clarifications. Architecture design often clarifies and defines non-functional requirements (NFR) or quality of services such as system reliability and performance. Refining or defining functional or non-functional requirements takes place when an architecture design is contemplated and the feedback from the design refines the original requirements after gaining consensus from stakeholders. Architecture rationales drive the feedback loop and the refinement process.

Refined requirements need to be mapped to AR as part of the ARM process. For instance, refinement of FR takes place when implementation issues are considered and system constraints or assumptions are clarified. An FR may have implications on, say, the usability of the system. With clarifications and tradeoffs, stakeholders may agree to change the requirement. The rationale of the refined requirements can then be recorded together with the AR mapping all relevant requirements.

NFR is an important part of architecture modeling because NFR need to deliver quality of services in order for FR in a system to be realized. NFRs are analyzed and defined in an iterative way in the architecture process [18, 19]. Agreed NFR are documented in the Business View. AR will document the rationale and map the defined or refined NFR to any relevant FR and design.

ARM uses AR as a catalyst to refine FR and NFR. Outcomes of the refinement process would be the refined requirements, FR or NFR, within the Business View. ARs are created in two categories:

- map inter-related requirements through AR, i.e. requirements to requirements
- map requirements to design components through AR, i.e. requirements to designs
4.2. Architecture Decomposition

The architecture development process is an iterative process of decomposing architecture designs. ARs are created during iterations of the decomposition process to cross reference or map architecture elements with increasing level of details. AR would map requirements to designs across various views and within the same view. Therefore, it is necessary for elements within views, at all levels, to be uniquely identifiable. During decomposition, the mappings could connect between requirements, between requirements and design components, and between design components. We define design components as components that belong to the Information View, the Computation View or the Technology View.

Figure 2 shows an example of architecture decomposition as more specific and detailed architecture designs are created for each of the requirements. The increasing level of details provided by the architecture design would eventually satisfy all architectural level requirements in the Business View.

Figure 2. Architecture Decomposition with AR

The ARM decomposition process goes through multiple iterations. It uses AR to map requirements to designs or to map designs of different levels. This rationale centric technique for decomposing designs into finer details has several merits:

- Ensure that the relationships between FR, NFR and the architecture designs are maintained
- Ensure that design decisions are rational
- Provide linkages for tracing design models between different views. Instead of tracing between requirements and design elements in most software engineering processes, ARM tracing could take place between any elements in business, information, computation and technology views.
- Serve as an aid for checking completeness and soundness of architecture model
For complete architecture traceability, it is necessary that all architectural level requirements in the Business View are mapped to architecture designs in the other views. Section 4.3 describes architectural level requirements and when architecture decomposition process is considered complete.

A network of ARs representing the design rationales of the architecture will be formed when the architecture design process is complete. These ARs are mapped to requirements and designs at different levels of details and so they themselves form a hierarchy. The richness of information contained in this network of ARs can provide a basis for evaluating architecture completeness, architecture coverage, impact analysis for design changes, and architecture maturity.

As the architecture designs are decomposed and designs with more details become available, it is expected that the level of risks, as represented by OCR and ICR, would decrease in the detailed ARs. This risk relationship between related ARs at different levels requires further studies and is potentially an area where architecture complexity and design behaviour can be further explored. If the acceptable risk is set at a lower value, say 10\%, then the level of details required in architectural designs in order to meet the risk criteria would increase.

### 4.3. Architecture or Detailed Design

Architecture is concerned with the structural design of the system. Its objectives, activities and outcomes are different to detailed software design. There is currently no agreed definition on the amount of details that are required in architecture modeling in order to satisfy architecture objectives. Instead of using Intensional and Non-local properties [13] to distinguish architecture activities, we propose a method for distinguishing architecture activities from detailed design activities based on the level of risks. The level of risks represents the uncertainty of the architecture model to meet its requirements. Architects would provide estimates for Outcome Certainty Risk (OCR) and Implementation Certainty Risk (ICR) at each decision point during requirements refinement and architecture decomposition. OCR represents uncertainty of meeting outcome objectives set out in FR or NFR. This risk may arise for a number of reasons:

- FR or NFR may be ambiguous and need to be clarified or redefined
- A design may have an impact on certain aspects of NFR and the extent of the impact is unknown
- Certain external environmental factors may have an impact on the implementation of requirements and the extent of the impact is unknown
- ICR represents uncertainty in implementing the design. This risk may arise for the following reasons:
  - Technical feasibility of the implementation
  - Uncertainty due to complexity of the design
  - Uncertainty due to lack of experience, knowledge or skills

These two types of risks can be resolved through iterations of refinement and decomposition. ARM provides a means to record and analyze these risks, and reduce them to the extent that both architects and stakeholders agree on the refined requirements and they are relatively certain that the architecture model would satisfy these requirements. OCRs and ICRs of higher level ARs are adjusted accordingly to reflect the risk levels of the new designs. When this certainty level is reached for all architectural requirements and designs, the architecture model is complete.
Different requirements in the system post different levels of risks. The level of design details required in various parts of an architecture model vary depending on the level of risk. For instance, a reusable item with a well defined behavior and interface does not require additional in-depth architecture design. On the other hand, requirements that are complex in nature or have extensive NFR implications may need further investigation into its technical feasibility or detailed design to ascertain feasibility and cost. Requirements that need architectural investigations are called architectural level requirements. Requirements that do not require architectural investigations because their risks are low become the concern of detailed system design activities.

OCR and ICR depict the levels of uncertainties of a design and represent the architect’s confidence level of the design meeting its requirements objectives. As such, they can also be used to determine whether further decomposition is required for the particular design area. Since the determination of OCR and ICR are semi-objective, the acceptable level of risk becomes a semi-objective assessment and could be set to a defined acceptable level depending on projects. If we use 20% as the acceptable risk levels, and both OCR and ICR are below 20%, it is said that 80% of the particular architecture design is free of major structural issues. Should either OCR or ICR be above this level, investigation or modeling at a more detail level is required until the risk of the decision is reduced to 20% level or below. The following are necessary conditions for the completeness of an architecture model:

- All known NFRs have been identified through requirements refinement
- All known architecture level requirements, FR and NFR, have corresponding mappings to design components through AR, directly or indirectly
- All design components have corresponding mappings through AR
- For each AR, OCR and ICR are below a defined acceptable level

All high risk FRs, as indicated by OCR and ICR, should be part of architecture level requirements and need to be a part of architecture design. Low risk FRs do not require architectural design because they do not have structural impact on architecture. NFR usually have higher risk factors because (a) NFRs usually affect a large part of the system; (b) NFRs are usually competing with other NFRs and (c) the impact of a design to satisfy an NFR requires more design attention. Therefore, all NFR should be considered in architecture modeling until OCR and ICR risk levels become acceptable. This mechanism provides a way to define the scope of architecture activities and allow measurement of completeness and soundness of architecture outcomes. A complete architecture model is defined as a model of a system, based on a set of requirements, which has a low possibility that subsequent detailed designs would have major impacts on the overall implementation, structure or outcome of the system.

5. A Retrospective Case Study

The case study originates from initial designs performed by the first author on an E-payment product. The prototype was built and then handed over to another team. The prototype was subsequently abandoned when the second team found that they could not enhance it to become a full product due to the lack of the system’s design knowledge. There was no AR to explain why the architecture was designed the way it was. Risks as represented by ICR were not considered at the time, and in retrospect they were exceedingly high because the second team did not have the background knowledge to continue with the development. The purpose of this case study is to illustrate what
additional information should be available in order to enable design rationale retention without original designers’ involvement. The system is highly complex and only a small part of the system is described here.

The main function of the E-payment system is to *truncate checks* at the point of deposit and use check images for central clearing. The prototype was developed using C++ and architecture specification and detailed design specification were also available.

Checks are deposited at bank branches and they need to be transported, or presented, to a central clearing house for sorting and then moved to paying banks to validate that they can be honored. The system uses the Paperless Automated Check Exchange and Settlement (PACES) standards set out by Financial Services Technology Consortium (FSTC) [20]. There are a number of general functional requirements of this system.

- Images and data of checks are bundled and transmitted in presentment files
- All transmissions of presentments are encrypted
- All presentments are digitally signed
- Checks data and images are bundled in a standard format and in this case, X9.37 and X9.46 format, for exchange
- Clearing cycles must be completed within their respective time windows
- Perform financial postings

For a COTS product, we have to make certain assumptions regarding usage. We initially assume that an average file contains 50,000 checks and the file size is about 6GB. The system should process a peak of one million checks within a ninety minutes window.

![Figure 3. Multi-pass Clearing Model](image)

---

**Business View**

<table>
<thead>
<tr>
<th>PR</th>
<th>NFR</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1.0 – Decompose Batches</td>
<td>R10.0 – Security</td>
</tr>
<tr>
<td>R1.1 – ECF Data in X9.37 Format</td>
<td>R10.1 – Encrypt data using PKI</td>
</tr>
<tr>
<td>R1.2 – Image Data in X9.46 Format</td>
<td>R10.2 – Authenticate file using Digital Signature</td>
</tr>
<tr>
<td>R2.0 – ECF and Image Processing</td>
<td>R11.0 – Performance</td>
</tr>
<tr>
<td>R2.1 – Reconcile ECP Data with Batch Header</td>
<td>R11.1 – Process 150 checks per second</td>
</tr>
<tr>
<td>R2.2 – Store ECP Data in Database</td>
<td>R12.0 – Reliability</td>
</tr>
<tr>
<td>R2.3 – Store Check Image in Image Store</td>
<td>R12.1 – Recovery from interruptions during processing</td>
</tr>
<tr>
<td></td>
<td>R12.2 – No loss of information from check batches</td>
</tr>
<tr>
<td></td>
<td>R12.3 – No duplicate entry</td>
</tr>
</tbody>
</table>

**Information View**

- Image Start
- ECP Data

**Computation View**

- PresentationFile
- Decryption(): void
- Authentication(): void
- Decomposition(): void

- Check/Clearing
- ProcessChecks
- Share(): void
- ReconCec(): void
- Decomposition(): void

**QaR**

- Assumptions:
  - 1st suitable model
  - Assumption of file size: 60K (560 MB takes 120 sec)
  - Constraints: disk transfer rate
  - Non-real processing logic
  - Risk: disk performance to serve computing processes
  - Strengths: ease of programming
  - Weakness: 3 passes of file take 36 sectors

---

SWINBURNE UNIVERSITY OF TECHNOLOGY

SUTIT-TR2004.05 – Architecture Rationalization: A Methodology for Architecture Verifiability, Traceability and Completeness

Prepared by: Antony Tang, Jun Han

Wednesday, September 1, 2004
There are two possible models to design the check clearing process in the system. The first model, depicted in Figure 3, uses a sequential method of processing where the presentment file is decrypted and authenticated by a security process. Clear text presentment is stored in an intermediate file. The intermediate file is then processed by a separate process to decompose the data into check images and check data. This model requires multiple passing of presentment files in encrypted and clear text formats. This model would involve three passes of data files using one write and two reads. This design requires 120GB of temporary space. This design would satisfy the functional requirements but we are not certain that it would satisfy the NFR, specifically the performance criteria under load. We estimate the ABI to be 6 and OCR to be 50% (medium risk). The implementation of the model will be simple and we estimate the ACI to be 5 (average cost) and ICR to be 20% (low risk). ER is therefore 0.5.

Figure 4 shows the single-pass clearing model. This model uses a single process to decrypt the presentment file and store it partially in memory through buffer management. Internal control is passed through the ProcessControl() module to decompose, authenticate and process in a non-sequential way. This model requires one pass of the original file and does not require use of temporary disk space. This model will satisfy FR and it is given an ABI index of 8.

![Figure 4. Single-pass Clearing Model](image)

Single-pass should perform better than the multi-pass model because it requires one read only, its OCR is given a value of 30%. The cost (ACI) and complexity of implementation (ICR) is higher and they are estimated to be 7 and 30% respectively. ER is estimated to be 0.61 at this stage.

The ER ratio is higher in the single pass model than the multi-pass model mainly because it has lower performance risk. Therefore, we continue to pursue the single pass model. A decision cannot be finalized until risk factors represented by OCR and ICR in the single pass model are reduced to an acceptable level of 20%.
For the Single-pass model to be acceptable, the performance levels must be met. Further assumptions on usage are required and feasibility tests are performed:

- We make an assumption that there are a minimum of 4 and a maximum of 20 bank connections during the check presentment window.
- Processing time of a single process for decryption and authentication using Elliptic Curve Cryptography on a single node is tested and the results show that it can process 50,000 checks within 10 minutes with six concurrent processes.
- It is tested that a single node (computer server) could support insertion of images and data into the data stores at a rate of 500 checks per sec.
- It is tested that reading a data file of 6GB requires approximately 120 seconds.

Based on the assumptions and the testing, we estimate that the Single-pass model can process up to 300,000 checks, or 36GB data, in about 40 minutes on a single node. Therefore, the OCR level can be reduced to an acceptable level of 20% if the architecture design uses two nodes for processing simultaneously.

![Decomposed Single-pass Model](image)

Figure 5. Decomposed Single-pass Model

This architecture requires NFR to be refined to reflect new reliability requirements due to a dual-nodes architecture. In Figure 5, AR2 is the new rationale for mapping the dual node system. It maps performance and reliability requirements in NFR to the Computation View and the Technology View. The Single-pass design requires AR3 to map the additional reliability requirement (R12.4) because of the introduction of dual-nodes. ICR in AR1 is still above the acceptable level and so more decomposition is required to reduce implementation risk. The risk lies in the management of buffers and control mechanism in a single process to eliminate multiple reads of external files. The single-
pass process has to (a) create and manage dynamic buffers that store the presentment file during processing; (b) the presentment is decrypted, authenticated and processed in a staggered and progressive way. AR5, therefore, contains the rationale of introducing RawBuffer and ECPBuffer to manage input as it is read and processed. AR4 contains the rationale of a checkpoint requirement for process recovery. AR5 maps a new data entity PresentmentFileCtl in the Information View and Checkpt component in the Computation View. ICR in each design area is now reduced to 15% after iterations of architecture refinement and decomposition. As sub-designs of AR1 are explored, its risks are subsequently reduced to an acceptable level and the architecture design in this area is considered complete.

Table 1 shows the elements traced by AR in each iteration of refinement or decomposition. More design details are captured as the architect considers the recovery aspects of the system through the design iterations. AR2 and AR3 map the requirements to the new design, including the introduction of the new nodes. A new requirement (R12.4) is created because of the new nodes. In the 3rd and 4th iterations of the design, more design decomposition of the PresentmentFile is performed and AR4 and AR5 are created to map the newly designed elements.

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>Iteration 2</th>
<th>Iteration 3</th>
<th>Iteration 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR 1</td>
<td>R1.0, R2.0, R10.0, R11.0, R12.0, PresentmentFile</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AR 2</td>
<td>R11.0, R12.0, PresentmentFile, Node1, Node 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AR 3</td>
<td>R12.4 (new), PresentmentFile, RecoveryProcess, Node1, Node2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AR 4</td>
<td>R12.1, PresentmentFileCtl, CheckPt (Note: AR1 depends on AR4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AR 5</td>
<td>PresentmentFile, ECPBuffer, RawBuffer (Note: AR1 depends on AR5)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Tracing Requirements and Design Elements through Design Iterations

Since new designs introduced through AR4 and AR5 are sub-designs of the PresentmentFile, which is justified by AR1, any changes in the risk estimation of AR4 or AR5 would affect the risk estimation of the original AR1. Iterations 3 and 4 shown in Table 2 reveal details of the risk estimation of AR4 and AR5. In iteration 5, AR1 is adjusted to reflect the lowering of risks because more certainties are ascertained through architecture designs in iterations 3 and 4. It should also be noted that AR3 is used to map the recovery requirements of the two nodes. The mechanism and the design to satisfy this requirement are yet to be specified. For instance, the fail-over of the two nodes is one of the issues remained to be addressed. Processes in a failed node should restart within a specified time after fail-over. It implies that another NFR will need to be introduced to refine the original requirements. Consequently, the risk of AR3 is still above the acceptable level, and further iterations of requirements refinement and architecture decomposition is required to reduce AR3’s risk to an acceptable level.
<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>Iteration 2</th>
<th>Iteration 3</th>
<th>Iteration 4</th>
<th>Iteration 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Choose single or multi-pass)</td>
<td>(Decide on 2 nodes)</td>
<td>(Recovery Process Refinement)</td>
<td>(Presentment Process Refinement)</td>
<td>(Reassess AR1)</td>
</tr>
<tr>
<td>AR 1</td>
<td>ABI=8; OCR=30%;ACI= 7;ICR=30%; ER=0.61</td>
<td></td>
<td>ABI=8; OCR=20%;ACI= 7;ICR=15%; ER=0.79</td>
<td></td>
</tr>
<tr>
<td>AAR 1</td>
<td>ABI=6; OCR=50%;ACI= 5;ICR=20%; ER=0.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AR 2</td>
<td>ABI=8; OCR=20%;ACI= 7;ICR=10%; ER=0.83</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AR 3</td>
<td>ABI=7; OCR=30%;ACI= 7;ICR=30%; ER=0.53 (More decomposition is required)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AR 4</td>
<td>ABI=8; OCR=10%;ACI= 7;ICR=20%; ER=0.85</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AR 5</td>
<td>ABI=8; OCR=20%;ACI= 7;ICR=15%; ER=0.79</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Assigning Quantitative Rationale Values through Design Iterations

The ARs, in this particular case, document the rationale of the system design which can be used to trace and verify the architecture. Previously, the need to record the architecture design rationale was not considered. As cross-teams development and development outsourcing become a common place, AR has a vital role to play in architecture development. The lack of AR and its traceability in the initial case inhibited the understanding of the original architectural design and so the team which took over the check truncation project judged it easier to redesign from the beginning. In general, the design information may be intuitive or obvious to architects involved in the design process but when the architects are no longer part of the project team, the intuitive knowledge become unavailable if not recorded. If ARM had been available, AR would have provided traceability in designs and requirements, and would have made it much easier to understand the original architecture.

6. Conclusion

In this paper, we have provided an Architecture Rationalization Method based on qualitative and quantitative reasoning. Using Expected Return (ER) in Architecture Rationale (AR), architecture decisions become quantifiable and verifiable. A key aspect of the quantification process is the consideration of risks in terms of expected outcome and implementation. The uncertainty, as represented by risks, becomes the driving force of the architecture refinement and decomposition processes. The qualitative and quantitative properties of AR and AAR provide the basis for verification of architecture choices. Architecture decisions documented in ARs map requirements and designs in architecture views to provide traceability of decisions.
The case study demonstrates that essential information for tracing and verifying architecture can be captured by ARM. This information is also essential in supporting the architecture evolution. A network of traceable ARs can also provide much information to support measurement of design complexity, design risks, design coverage and metrics to measure performance. All these areas warrant further studies.

A limitation to any architecture design might be the incomplete quantifiable information for rationale analysis. ARM circumvents this by guiding architects to consider risks based on the incomplete information. Documented ARs provide metrics for verification of decisions and retrospective assessment of architects' judgments. We will survey the risk assessment capabilities of architects in a follow up research to further study acceptable risk levels. ARM can be implemented by an automated tool using UML notation to facilitate the traceability of architecture models. We are currently in the process of extending TRAM [11] to provide ARM capabilities.
7. References


