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

Antony Tang Jun Han
Faculty of ICT,
Swinburne University of Technology,
Melbourne, Australia
E-mail: {atang, jhan}@it.swin.edu.au

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. 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), which is based on architecture rationale, 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 not many studies and practical methods to verify architecture designs, to trace design rationales, or to ensure the architecture model 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 rationalization 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 and then abandoned due to the lack of design rationale. 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 behind the architecture decisions have vaporized [4] and could not be traced or verified. Vital information such as constraints, assumptions, considerations and tradeoffs, which are essential to the understanding of the architecture, are missing.

Given the issues underlying architecture designs, we analyze key elements in architecture modeling in an attempt 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. Although detailed design is a continuum of architecture design, architecture activities are distinct from detailed software 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 distinction between the two activities.

Our motivation is to represent architecture rationale and incorporate it into the architecture process to address the issues of verifiability, traceability and

Proceedings of the 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'05)
0-7695-2308-0/05 $20.00 © 2005 **IEEE**

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 elements being traced.

Architecture rationale has been identified by researchers and standards bodies as crucial in the architecture design process [4, 16, 18, 20], but none of the architecture frameworks surveyed [22] 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 [16]. Our contributions in this paper are the following:

- Introduce the Architecture Rationalization Method (ARM) as a framework to systematically rationalize and record architecture decisions.
- Make use of the Architecture Rationale (AR) for recording qualitative and quantitative rationales to facilitate architecture *verifiability.*
- Provide requirements to requirements linkages, requirements to design linkages and design to design linkages through AR for architecture *traceability* of decisions and for the representation of design reasoning.
- 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 and reliability that cut-across the design of a system. 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. These architecture views not only include software architecture but encompass requirements and business strategies, data models and supporting infrastructure as well. Different architecture frameworks (AF) such as RM-ODP [17], FEAF [7], TOGAF [23] and DoDAF [9] prescribe slightly different viewpoints for architecture modeling. The following are a generalization of the viewpoints:

- Business / Enterprise View
- Information / Data View
- Computation / Application View
- Engineering / Technology View

Different views represent architectural designs from different perspectives. Models contained in views are the results of design considerations, tradeoffs and decision making. In this paper, architecture refers to all architecture elements using these generalized viewpoints with architecture rationales cross-cutting these viewpoints to provide connectivity of decisions.

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 design specification during the design phase. 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 [14]. There are a number of research work related to architecture design rationale, architecture evaluation and the application of economic considerations [1, 2]. These methods require the presence of architects and stakeholders to provide the information. 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) or

- A design can be justified and 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.

2.3. Traceability

An architecture model, as represented by views, is usually documented without cross referencing between system requirements and design components. Architects would consider business requirements and system requirements in order to select an appropriate design from different choices by using reasoning, experience and intuition. A study by Ramesh and Jarke [21] indicates that the traceability focus of the surveyed systems are mostly on the satisfaction of requirements. The lack of traceability of requirements and designs to rationale inhibits the verifiability of architecture design.

It is noted in [13], that 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 also recommended by the IEEE standards [14, 15]. In our case study, the lack of traceability has little impact on initial implementation of a system, but it inhibits another team of designers to properly understand why the design is done in a given manner.

2.4. Differentiating Architecture and Detailed Design

Perry and Wolf [20] 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" [16]. The obvious questions are [13, 22] :

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

There has been some suggestions in this area [10] but our survey shows that there is no distinction between architecture and detailed design made by any of the architecture frameworks [22]. The lack of understanding of this area implies that (a) architecture design activities could go beyond what is necessary, i.e. over architecture; (b) the architecture design could be incomplete; (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.

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 linkages, between elements in an architecture model through an AR.

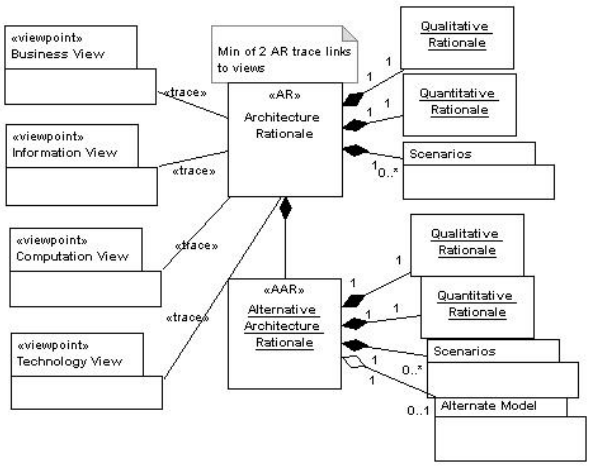

Figure 1. Architecture Rationale Model

The AR model is an UML representation depicted in Figure 1. Both 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 linked, 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.

Alternate Architecture Rationale (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 to 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 object that documents the rationale of a design choice. Elements of AR contain the following:

- An identifier to uniquely represent the AR
- At least two or more links to elements contained in the architecture view model. The linked elements may be from the same view or from different views. Each element in a view is uniquely identifiable
- Qualitative rationale (see below)
- Quantitative rationale (see below)
- Scenarios to depict constraints or special cases
- AR is associated with zero or more AAR and AAR contains alternative design 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. As such, recording architecture rationale coupled with the ability to trace design decisions may be a potential solution to address architecture erosion and drift issues raised by Perry and Wolf [20]. AR and AAR make use of qualitative rationales suggested in ATAM [2] 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. Design strategies or principles that dictate the decision, such as modularization, abstraction or separation of concerns, can be documented under this category.

Architects could use some or all of QuR categories depending on the context of the architecture decision. For instance, AR may link a business requirement to a computational component, and then link 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 to meet the requirements. Similar assessments would be made for alternative designs. CBAM provides a method to calculate the benefit score [1] that is useful for ranking important decisions but the cost of having a group of stakeholders to constantly vote for all architecture decisions could be high. *ABI* provides an alternative 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 risk or the uncertainty level of the architectural design meeting 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 three merits (a) architecture rationale is quantified at each decision point; (b) *ER* takes into account risks or uncertainties in architecture modeling; (c) the soundness of each architecture decision can be assessed and measured for evaluation and verification. The use and interpretation of these quantitative elements are different to the existing quantitative models [1, 3] in that QaR requires architects to make estimates for these elements. However, we believe that experienced architects would provide consistent estimates because the basis of these estimates should be a well supported conscious decision of tradeoffs between requirements, designs choices and risks.

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

$$EB = (1 - OCR) * ABI$$

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

$$EC = (1 + ICR) * ACI$$

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.

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

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 [5]. As such, we argue that it should be done as an ongoing and incremental activity during the architecture construction process. Some research work suggests that software architecture risk is evaluated when a model is relatively complete [12], but ARM risk analysis is carried out during the entire decision making process at each decision point. An automated tool would facilitate the recording and calculation of quantitative rationale for practical use.

4. Architecture Rationalization Method

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

- A requirement refinement technique
- An architecture decomposition technique

Outcomes of ARM are architecture rationale (AR) that link 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 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 FR or NFR takes place when an architecture design is contemplated and the feedback from the design refines the original requirements. Architecture rationales drive the feedback loop and the refinement process.

Refined requirements are linked to AR as part of the ARM process. An FR may have implications on, say, the usability of the system. With clarifications and tradeoffs, stakeholders may agree to change the requirement. 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 [6, 8, 19]. Agreed NFR are documented in the Business View. AR will document the rationale and link 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 to link inter-related requirements and design components.

4.2. Architecture Decomposition

The architecture development process is an iterative process of decomposing architecture designs to realize the implementation of the system. ARs are created during iterations of the decomposition and decision making process that cross reference or link architecture elements with increasing level of details. During decomposition, the linkages 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.

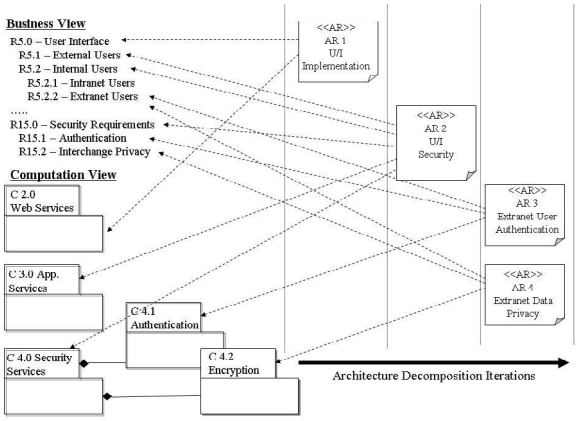

Figure 2. Architecture Decomposition with AR

The ARM decomposition process goes through multiple iterations. It uses AR to link requirements to design or to link 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 that cross-cut architecture views are maintained

- Ensure that design decisions are rational
- Provide linkages for tracing design models
- 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 linked to architecture designs in the other views. Section 4.3 describes architectural level requirements and when architecture decomposition process is considered complete.

4.3. Architecture or Detailed Design

Architecture is concerned with the structural design of the system. Its objectives and activities are different to detailed software design. The need for such distinction lies in the role of architects and the need for architecture verification. 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 [10] 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. 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 stakeholders 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 in meeting its objectives. So they can 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. We choose to use 20% as a guideline. This guideline is arbitrary in which the authors are comfortable with, more empirical work is required to establish a standard. Should either OCR or ICR be above this level, investigation or modeling at a more detail level is required until the risk 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 linkages to design components through AR, directly or indirectly
- All design components have corresponding linkages 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 cut-across 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. This mechanism provides a way to define the scope of architecture activities and allow measurement of completeness 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 were not represented by ICR, and in retrospect they were exceedingly high because the second team did not have the background knowledge of the initial development. The purpose of this case study is to illustrate the design rationale retention that would be captured without original designers' involvement. The system is highly complex and only a small part of the system is described.

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++. Architecture specification and detailed design specification were also available.

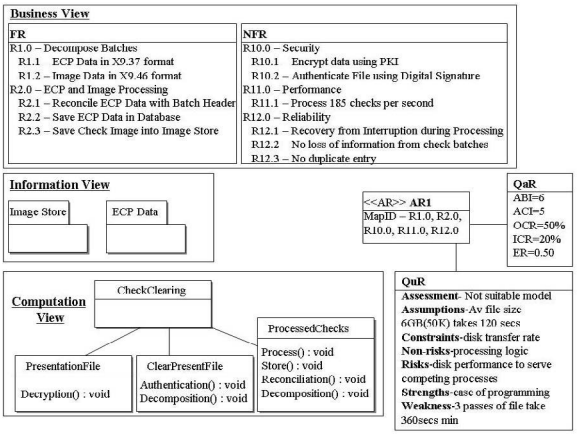

Figure 3. Multi-pass Clearing Model

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) [11]. There are a number of general functional requirements.

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

We 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.

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 and then processed separately to decompose the data into check images and check data. 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.

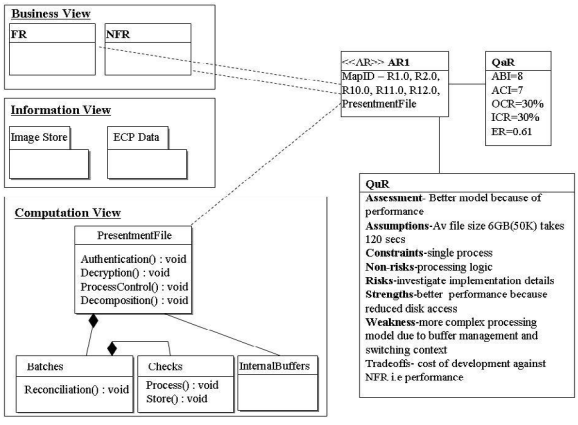

Figure 4. Single-pass Clearing Model

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 managed by ProcessControl() to decompose, authenticate and process in a non-sequential way. This model is given an ABI index of 8. It should perform better than the multi-pass model because of reduced processing, its OCR is given a value of 30%. ACI and ICR are higher and they are estimated to be 7 and 30% respectively. ER is estimated to be 0.61.

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%. Further assumptions and feasibility tests are required:

- An assumption of between 4 to 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 estimated 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 estimated 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 20% if the architecture design uses two nodes for processing simultaneously.

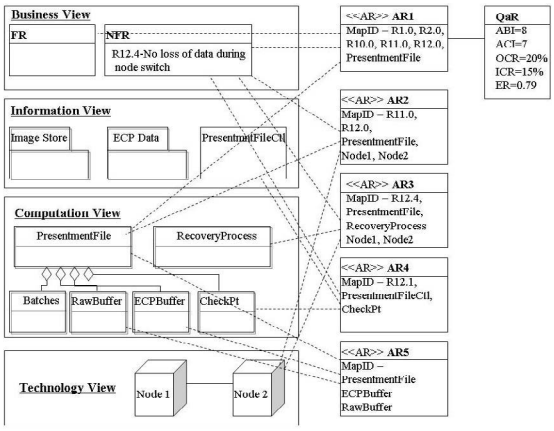

Figure 5. Decomposed Single-pass Model

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 linking the dual node system. It links performance and reliability requirements in NFR to the Computation View and the Technology View. The Single-pass design requires AR3 to link 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 links a new data entity *PresentmentFileCtl* in the Information View and *Checkpt* component in the Computation View. ICR in each design area is 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.

ARs document the rationale of the design for traceability and verifiability of architecture designs and decisions. Previously, 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 this 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. Design information may be intuitive or obvious to architects involved in the design process but when these people are no longer part of the project team, this knowledge vaporizes if not recorded.

6. Conclusion

In this paper, we have provided an initial framework of Architecture Rationalization Method based on qualitative and quantitative reasoning. Using Expected Return (*ER)* in Architecture Rationale (AR), architecture decisions become quantifiable and verifiable. A number of new ideas are incorporated into ARM. Firstly, the uncertainty, as represented by

risks, defines the *completeness* of the architecture *refinement* and *decomposition* processes. Secondly, qualitative and quantitative properties of AR and AAR provide the basis for *verification* of architecture decisions. Thirdly, AR and its linkages to architecture elements such as requirements and designs in architecture views provide *traceability* of decisions.

The case study demonstrates that essential information for tracing and verifying architecture can be captured by ARM. This information is essential in supporting architecture evolution. The lack of quantifiable information for rationale analysis is circumvented by ARM using quantitative rationale. Documented ARs provide metrics that can be used for verification of decisions and assessment of architects' judgments.

This research is work in progress and we are extending our research into ARM in a number of areas: (a) a semantic model of AR and its implementation using an existing UML tool, (b) an architecture decision graph and its inference capabilities to support complexity and impact analysis, (c) measurement of architecture design risks, design coverage and model accuracy based on the decision representation, and (d) an empirical survey of architects' risk assessment to further study acceptable risk levels in architecture design.

7. References

[1] J. Asundi, R.Kazman, and M. Klein, "Using Economic Considerations to Choose Amongst Architecture Design Alternatives," Carnegie Mellon University, Software Engineering Institute CMU/SEI-2001-TR-035, ESC-TR-2001-035, 2001.

[2] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Boston: Addison Wesley, 2003.

[3] B. W. Boehm, *Software Engineering Economics*. New Jersey: Prentice Hall PTR, 1981.

[4] J. Bosch, "Software Architecture: The Next Step," presented at Software Architecture: First European Workshop, EWSA 2004, St Andrews, UK., 2004.

[5] R. Charette, *Software Engineering Risk Analysis and Management*. New York: McGraw-Hill Book Company, 1989.

[6] L. Chung, B. Nixon, E. Yu, and J. Mylopoulos, *Non-functional requirements in software engineering*. Boston: Kluwer Academic, 2000.

[7] CIO-Council, "Federal Enterprise Architecture Framework version 1.1," 1999, http://www.cio.gov/archive/fedarch1.pdf.

[8] M. Denford, J. Leaney, and T. O'Neill, "Non-Functional Refinement of Computer Based Systems Architecture," presented at The 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems., 2004.

[9] Department of Defense, "Department of Defense Architecture Framework Version 1.0 - Vol 1 Definition & Guideline and Vol 2 Product Descriptions," 2003, http://www.aitcnet.org/dodfw.

[10] A. Eden and R. Kazman, "Architecture, Design, Implementation," presented at International Conference for Software Engineering, 2003.

[11] FSTC, "Paperless Automated Check Exchange and Settlement (PACES)," 2000, http://www.fstc.org/projects/paces/index.cfm.

[12] K. Goseva-Popstojanova, A. Hassan, A. Guedem, W. Abdelmoez, D. E. M. Nassar, H. Ammar, and A. Mili, "Architectural-level risk analysis using UML," *IEEE Transactions on Software Engineering*, vol. 29, pp. 946-960, 2003.

[13] J. Han, "TRAM: A Tool for Requirements and Architecture Management," presented at Proceedings of the 24th Australasian Computer Science Conference, Gold Coast, Australia, 2001.

[14] IEEE, "IEEE/EIA Standard - Industry Implementation of ISO/IEC 12207:1995, Information Technology - Software life cycle processes (IEEE/EIA Std 12207.0-1996)," IEEE 1996.

[15] IEEE, "IEEE/EIA Guide - Industry Implementation of ISO/IEC 12207:1995, Standard for Information Technology - Software life cycle processes - Life cycle data (IEEE/EIA Std 12207.1-1997)," IEEE 1997.

[16] IEEE, "IEEE Recommended Practice for Architecture Description of Software-Intensive System (IEEE Std 1471-2000)," IEEE Computer Society 2000.

[17] ISO/ITU-T, "Reference Model for Open Distributed Processing (ISO/ITU-T 10746 Part 1 - 4)," Information Standards Organisation 1997.

[18] J. Lee and K. Lai, "What is Design Rationale?," in *Design Rationale - Concepts, Techniques, and Use*, T. Moran and J. Carroll, Eds. New Jersey: Lawrence Erlbaum, 1996, pp. 21-51.

[19] M. Moore, R. Kazman, M. Klein, and J. Asundi, "Quantifying the Value of Architecture Design Decisions: Lessons from the Field," presented at International Conference on Software Engineering, 2003.

[20] D. E. Perry and A. L. Wolf, "Foundation for the Study of Software Architecture," *ACM SIGSOFT Software Engineering Notes*, vol. 17, pp. 40- 52, 1992.

[21] B. Ramesh and M. Jarke, "Towards Reference Models for Requirements Traceability," *IEEE Transactions on Software Engineering*, vol. 27, pp. 58-93, 2001.

[22] A. Tang and J. Han, "A Comparative Analysis of Architecture Frameworks," presented at 1st Asia-Pacific Workshop on Software Architectures and Component Technologies, APSEC 2004, Korea, 2004.

[23] The Open Group, "The Open Group Architecture Framework (v8.1 Enterprise Edition)," 2003, http://www.opengroup.org/architecture/togaf/#download.