Modeling Architectural Non Functional Requirements: From Use Case to Control Case

Joe Zou and Christopher J. Pavlovski
IBM Corporation
55 Pyrmont Street, Sydney
NSW Australia
joezou@cn.ibm.com, chris_pav@au1.ibm.com

Abstract

While the functional requirements of a system can be effectively modeled through the Use Case driven approach, there is no standard or de facto method for modeling non-functional requirements of the system architecture. Often such requirements are dealt with in a reactive manner rather than proactively. Yet increasingly a contributing factor in project difficulty and failure are the non-functional requirements imposed on the solution architecture. This paper proposes a Control Case approach to record and model non-functional requirements. This technique enables the control case to represent the non-functional requirements from different perspectives, most typically the various operating conditions. Furthermore, we propose an extension to the "4+1" view model for depicting software architecture by adding the control case view. The combination of both the use case and control case views thus reflects the complete requirements across the collective system life cycle views: design, process, implementation and deployment.

1. Introduction

In general terms, functional requirements identify what a system is intended to do. Conversely, Non-Functional Requirements (NFRs) articulate the set of parameters for system behavior. This broadly includes requirements such as system performance, reliability, security, response time, quality of service, and system availability [1]. NFRs are also known as technical requirements, system properties, or quality attributes.

Clements and Northrup suggest that whether or not a system will be able to exhibit its required quality attributes (NFRs) is largely determined by the time the architecture is chosen [2]. Furthermore, performance requirements are often ignored until the end of the development cycle such as integration testing [3].

Traditional IT systems largely dealt with transaction loads originating from within the organization, accommodating demands generated by hundreds or perhaps thousands of users. The evolutionary expansion that the Web has brought, means that a more demanding set of non-functional requirements are imposed on e-Business solutions. Whilst the expected response times, reliability, and availability may have remained largely constant during this e-Business transformation, the magnitude of users has increased significantly; from several thousand often into the millions of users.

In view of the growing transactional demands upon e-business solutions, the non-functional aspects of an architecture are increasingly important, motivating the need to manage these requirements throughout the life cycle of systems development.

In this paper we propose a complement to use cases in order to supports modeling of NFRs. We introduce the notion of Control Case, used to record an NFR, and associate these with a use case via an operating condition. The proposed control case technique places a focus on these critical requirements and facilitates the management of NFRs throughout the development life cycle. In a formal sense, we also outline how NFRs, recorded as control cases, provide systems coverage by extending the UML 4+1 view model of software architecture [4].

Generally, NFRs are identified as a distinctly separate exercise to the capture of functional requirements. Through a clearer association of non-functional to functional requirements, it is hoped that these techniques will further support the capture and management of NFRs throughout the systems development life cycle.

IEEE International Conference on e-Business Engineering (ICEBE'06)
0-7695-2645-4/06 $20.00 © 2006 IEEE

2. Background and Related Work

Since the introduction of the use case by Ivar Jacobson in 1986 [5], the notation has become a ubiquitous tool for IT projects. Use cases have been applied in a variety of ways including the capture of functional requirements, planning task breakdown and required resources, performing domain analysis, creation of object and component models, designing test cases, and estimation [6].

While the functional requirements of the solution has received much attention in the literature, the non-functional aspects of an architecture have received considerably less interest, with no clear standard or tool emerging. The non-functional requirement is inherently non-behavioral in nature. Hence, a reason that the use case model is not intuitively suitable for addressing these non-functional characteristics is that its purpose is "to define a piece of behaviour of a classifier (including a subsystem or the entire system) without revealing the internal structure of the classifier" [7]. One example in addressing this divide is illustrated by the Rational Unified Process (RUP). Moreover, the capture of requirements that are not addressed by the use case model is achieved using an artifact called the supplementary specification [8]; normally used to define the system-wide, non-behavioral requirements.

2.1. Significance of the NFR

A crucial phase in the software development life-cycle is the capture of requirements. Whether the approach is iterative or more traditional in nature, the capture of all requirements is essential in defining the boundaries and capabilities of the system to be developed. Project failure has traditionally been attributed to a range of factors, in particular to the effective capture of functional requirements. Alan *et al*. observe that 'requirements errors typically comprise over 40% of all errors in a software project' [9].

More recently, the importance of concise NFRs and its relatedness to project failure is becoming more apparent. For example, the consequences of performance failure vary considerably, and may include: damaged customer relations, business failure, lost income, project cost increase due to additional resources required to resolve the overlooked performance requirement, reduced competitiveness, and ultimately may contribute to project failure [3]. Saleh and Al-Zarouni conclude that unlike functional requirements, NFRs are very hard to elicit [10]. Furthermore, once defined they are difficult to address in projects and are often referred to as soft goals [1].

2.2. Operational Vs Non-Operational NFRs

Non-functional requirements may be defined within two general categories. The first is from an operational perspective, these are defined as a Service Level Requirement (SLR). Examples are performance, response time, availability, integrity, security, scalability and recoverability. An alternative classification is the non-operational category, for example, the prescribed architecture and technology standards, portability, and maintainability. These later NFRs are not distinctly associated with a specific use case, rather, they are characteristics of the entire system to be developed.

In general, all NFRs appear to be viewed as system-wide qualities [1]. This means that these requirements are typically identified during a distinctively separate task to the recording of functional requirements, and the approach of separate analysis is strongly argued [11]. Such an approach however, omits an opportunity to capture the importance of certain NFRs when it may be clearly related to a functional requirement.

System architecture must address both functional and non-functional requirements. Otherwise the architecture will not be sustainable. Arguably, poor quality non-functional attributes of an architecture represent more risk to the business. This is because in today's business environment, more and more business operations heavily rely upon IT systems that are connected across corporate boundaries. On the other hand, more and more enterprises outsource their IT operations to service providers. As a result, meeting SLRs and Service Level Agreements (SLA) becomes the most critical concern in architecting today's IT solutions.

This motivates the need for a technique that can specifically address the non-functional aspect of a system. This paper proposes a control case concept, which is to define a specific quality requirement of a system, subsystem or component. A control case captures the SLRs, or operational NFRs, that the system needs to meet under specific operating conditions. For non-operational NFRs, the control case will document the constraints or possible control method through policy, process or procedures to ensure that NFRs are met accordingly.

2.3. Related Work on Use Cases

There is some work in applying use cases to address the non-functional requirement. Schneider and Winters suggest to add a special requirements section to the use case [30]. Armour and Miller discuss how one may document the 'non-behavioural requirements' using a

use case, immediately after the Alternative Flows and Exceptions section [12]. In [13], use-case modeling of requirements that relate to fault tolerance is detailed. The authors outline how to structure conventional use cases in order to capture fault tolerance requirements, suggesting that the process also forces early consideration of fault tolerance in the development process.

Other authors have pointed out that the inclusion of NFRs within the use case is distracting, making them harder to read [31]. However, the authors also observe that there are benefits in a close association of NFRs to a use case, suggesting the addition of extra fields to hold auxiliary information. Furthermore, Cockburn outlines that optional use case sections may contain entries such as priority, performance target, and frequency [32].

In [14] a prototype CASE tool that supports the representation of NFRs' is described. Their framework provides a *method catalog* and *correlation catalog* for decomposing and interrelating NFRs, more broadly defined as softgoals in their work. This work is more focused on managing a set of NFRs.

The idea of an Abuse Case is presented [15], which extends the use case to define a security requirement. Moreover, an abuse case defines a scenario where the results of the interaction are harmful to the system, actors, or stakeholders. This negative use case scenario is also applied by Sindre and Opdahl in defining a Misuse Case for functional requirements [16], while Alexander applies the Misuse Case [17], in part, to capture NFRs, notably the security requirements.

The Goal-oriented Requirements Language (GRL) is a formal language for modeling requirements, in particular NFRs [18]. The approach is based upon use case maps, and the focus is upon the goal rather than scenario-based specification of the NFR. An additional modeling language is also proposed in [11], where an aspect oriented approach to modeling NFRs is proposed. The objective of their work is towards validation of software architecture with respect to the defined NFRs.

The related work has focused on extending use cases, formal languages, or goals-oriented approaches to defining NFRs. In this paper an alternative approach that complements the use case, by demonstrating an association by an operating condition, is outlined. The control case is a scenario-based specification of the NFR and it is hoped that such techniques will facilitate capture of NFRs during requirements gathering by associating the NFR with its functional counterpart.

3. Modeling Architectural Requirements

The underlying principles of a control case are now outlined, describing how this is defined as part of non-functional requirements gathering. The objective is to define and outline how the control case is used in practice. Several methods exist for providing prescriptive processes to identify NFRs. This includes process oriented frameworks [19], Dobson's logical models of non-functional requirements [20], Kotonya and Sommerville's view points based framework [21], and Loucopulos & Karakostas comparable approach [18]. These and other methods may be applied as required by the underlying method applied by a project.

While the functional requirements model may consist of a use case model, interface descriptions and a problem domain model [22], the non-functional requirements model can be represented with a control case that captures the operating condition defined by the control case, see Figure 1 for a high-level definition.

CONTROL CASE: Name of the control case
Operating condition: Name of the operating condition.
Description: Description of the risk as a result of the operating condition and the focus area in mitigating the risk.
NFR Category: Category of the non-functional requirement.

Figure 1. Control Case Template

Use case modeling commences with identifying actors and discovering conceptual use cases [12]. Conversely, control case modeling starts with identifying the possible operating conditions (the operating condition association is defined in section 4.1) and the associated risk to the system stakeholders, mainly system owners and users. The conceptual control case can be discovered in each of the operational categories of non-functional requirements such as performance, availability, integrity, scalability, security and recoverability.

Fundamentally, a control case represents a set of quality statements that the system must meet in order to manage the risk exposed to system stakeholders such as owners and users. The risk may originate from low quality of services under different operating conditions. For example, the business may be at risk of:

- customer retention if the system is running under poor performance or response times;
- loss of revenue if the system does not provide sufficient availability of its services; or

- security attacks if the system security is poorly designed.

Figure 1 illustrates the template of the conceptual control case; an example of a completed conceptual control case is shown in Figure 2.

CONTROL CASE: Control response time
Operating condition: Concurrent user load.
Description: Under multiple concurrent user load, the system response time may become unacceptable to end users which leads to the risk of losing customers for the business. This control case defines the maximum response time for various transactions that the system must meet in order to mitigate the risk.
NFR Category: Performance and Capacity.

Figure 2. Completed Control Case

An iterative and incremental process is needed for the development of control cases. The initial conceptual control case can be further elaborated to create a detailed control case. The objective of the control case is to reduce the risk to a level where the system owners and users can accept the system and operating conditions. In practice, the elaboration of the conceptual control case will largely follow a risk assessment process, such as defined by National Institute of Standards and Technology (NIST). NIST proposes a nine-step process for risk assessment [23], see Table 1.

Table 1. NIST Risk Assessment Process

1. System characterization
2. Threat identification
3. Vulnerability identification
4. Control analysis
5. Likelihood determination
6. Impact analysis
7. Risk determination
8. Control recommendations
9. Result documentation

Although the NIST process is mostly adopted in risk assessment for IT security, the idea can be applied to risk assessment for IT performance, availability, scalability, integrity and recoverability. The objective is to determine the likelihood of the threat, the impact on the business, and the additional SLRs, or rather NFRs, that need to be defined for the system in order to mitigate the identified risk. In the control case context, the controls are equivalent to the methods used to achieve the SLR. The SLR is the 'what' and the control is the 'how'. Initially, the control case should focus on what needs to be achieved. The controls (how to achieve the SLR) can be addressed in the design stage.

The likelihood and impact can be rated at a scale of high, medium, low; alternative measurements may be applied as appropriate for the business situation. The risk level can be defined as the product of the likelihood and the impact in a risk-level matrix [23]. Shown in Figure 3 is the template of the detailed control case, see Appendix I for a completed example.

CONTROL CASE: Control response time	
Control Case ID: Unique identifier for control case.	
Operating condition: Name of the operating condition.	
Description: Description of the risk as a result of the operating condition and the focus area in mitigating the risk.	
NFR Category: Category of the non-functional requirement.	
Associated Use Cases: The related use cases.	
Regulatory Constraints: The non-functional requirement as a result of regulatory constraint, e.g. Sarbanes Oxley compliance.	
Business Constraints: The non-functional requirement exposed by business; e.g. business policy.	
Technical Constraints: Architecture or technology standards; e.g. J2EE compliance.	
Vulnerability: A weakness that can be accidentally triggered or intentionally exploited [23]; e.g. operating system security vulnerability.	
Threat Source: Any circumstance or event with the potential to cause harm to an IT system [23]; e.g. power outage.	
Operating Conditions	**Control Target (SLR)**
Operating condition scenarios that need to be managed; e.g. peak load.	The quality target that the system needs to deliver.
Impact if SLR not met: Description of impact if the SLR is not achieved, e.g. the business will lose 20% of its customers to competitors.	
Impact Rating: Impact grade: High, Medium, or Low.	
Likelihood: Likelihood the impact occurs: High, Medium, or Low.	
Risk Rating: Risk rating if Service Level Response (SLR) is not met: High, Medium, or Low.	
Controls: Methods to achieve the SLR.	
Residue Risk: The risk rating if SLR is met.	

Figure 3. Detailed Control Case

The control case provides an overview of the risk exposed to the system's stakeholders, and the service level requirements that need to be achieved in order to mitigate the risk. The control case can serve as the base for service level requirement definition and subsequently the service level agreement negotiation. Secondly, it can also be used in planning project activities, resources, cost and timeframe. Further more,

it can be applied to model the technical design and manage implementation of controls in later stages.

A more complete approach to constructing the control case may be defined by following the Software Performance Engineering (SPE) process [3]. In this approach the initial step is to assess performance risk, followed by identification of critical use cases. In the control case context the first step involves defining the high-level conceptual control case, with further refinement conducted as the 'identify critical use case' (i.e. control case) step.

4. Semantic Implications and Notation

From a metamodel perspective, UML presently does not represent the concept of 'quality of service'. Therefore, it may be argued that the non-functional requirements may not be effectively modeled using UML. There are some attempts in using UML extension mechanism, such as stereotype, constraint, and tag value pair to model performance; as advocated in OMG UML profile for schedulability, performance and time [24]. Nevertheless, this may not be applied in a straightforward manner to the broader scope of non-functional requirements such as scalability, availability, security and recoverability; due to the limitation of the meta-model.

Another possible approach is to apply UML's constraint concept to represent non-functional requirements. The UML constraint is a semantic condition, or restriction, expressed as a linguistic statement in some textual language [7]. A UML constraint can be expressed using any language, even a natural language as long as they are inside braces '{}' [25]. UML provides a formal language, which is the Object Constraint Language (OCL). OCL is based on predicate calculus, which is difficult to understand for people without good mathematical knowledge. In OCL, a constraint is a restriction on one or more values of, or part of, an object oriented model or system [26]. The definition implies that a constraint is applied to an object's state or behaviour, which is still limited to a functional aspect of the architecture. On the other hand, using natural language to express a constraint may lead to misinterpretation due to the ambiguous nature of natural language.

Some methods have recognized this limitation of the UML metamodel. For instance, the proposed IBM Architecture Definition Specification (ADS) extends UML metamodel to cover the non-functional aspect of architecture [27]. The refined metamodel includes NFRs and other operational model elements such as IT system, zone, location, and walkthrough. The non-functional requirement concept links to a model element, which specifies a quality requirement or constraint on the model element. The ADS model covers both functional and operational aspects of architecture and it is able to demonstrate their linkage through the *functional-operational-integration* package. An ADS model suggests that a node's location instance must satisfy the non-functional requirements. However, the model does not precisely define how the non-functional requirements are realized. This paper suggests that through control case modeling, the realization of the non-functional requirements can be effectively modeled from an architecture perspective.

4.1. Notation

We focus our attention on an introduction of the control case concept, hence this paper will only highlight the implications on the metamodel but will not go further on semantics discussion, (as most practitioners are not focusing upon the metamodel). From a semantic perspective, two new concepts require introduction to cater for control case modeling. The first is the operating condition and the second is the actual control case. Each concept is a kind of classifier in UML's terminology or type as defined in ADS [27].

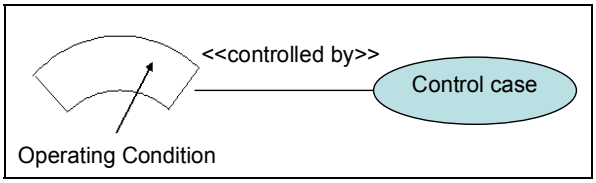

Figure 4. Control Case Notation

An operating condition represents the specific environmental status in which the IT system, subsystem or component is operating. This can be the load condition, bandwidth condition, or physical conditions such as network connectivity, power outage or even site disaster condition.

The control case defines the quality statements that the IT system, subsystem or component needs to meet under the specific operating conditions in order to match the business risk profile. Figure 4 illustrates the notation of an operating condition which is controlled by a control case.

In order to identify and associate a control case with a use case, during requirements gathering the analyst would therefore ask of each use case, 'whether any operating condition exists?' Traditionally, once the functional requirements have been extracted, then attention is drawn to NFR capture independently; if they are treated proactively at all. Identification of control cases during functional requirements gathering is a more natural way of extracting NFRs, due to the

direct association, and can assist in qualifying the significance of a particular NFR (by its association with a critical use case). Hence the control case approach seems to fit naturally with an evolutionary gathering of requirements, by close association with the particular use case. Figure 5 illustrates a final notation of a control case in association with an identified use case.

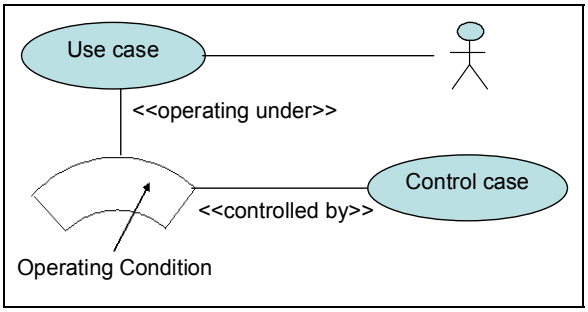

Figure 5. Relationship between Use Case and Control Case

4.2. The 4+1 View and Control Cases

The UML 4+1 process view clearly states coverage for NFRs, however present use case modeling does not intuitively support this. The proposed control case provides this necessary coverage; hence, we formalize the UML notation to cater for control cases. We suggest an extension to the traditional 4+1 UML view with the addition of a control case view to present a more compete picture of the systems architecture.

In the 4+1 view, the process view is intended to address the performance, scalability and throughput [4]. However, in practice it is difficult to model the quality aspect of architecture using UML; since the UML metamodel only supports the behaviour and structure concepts. This may present a risk that the non-functional requirements either get overlooked, or the realisation of the non-functional requirements is deficient or inadequate, since the architecture model does not have a clear representation of the non-functional aspect. Hence the control case view may be used to augment the 4+1 view. Together with the use case view, the control case view highlights that in each view the designer needs to consider both the functional and non-functional requirements to present a complete picture of the architecture.

Based on Booch's refined view model architecture [29], proposed by Kruchten [4], the diagram below adds the control case view to present a 4+2 view of architecture, see Figure 6.

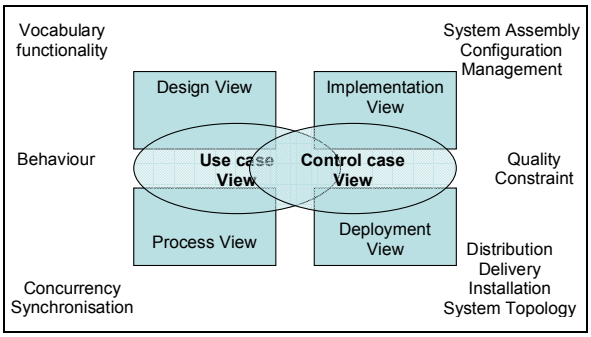

Figure 6. Modelling System Architecture with 4+2 Views

We can see from the 4+2 diagram, the control case view captures the non-functional requirements under different operating conditions. Thus when architects and developers prepare the design, process, implementation and deployment views, this implies that they draw input from both the use case views for functional requirements, and also the non-functional requirements from the control case views.

In order to judge whether it is necessary to introduce the control case view to the system architecture model, we can refer to Teeuw's six general quality criteria for a conceptual model [28]:

1. Completeness,
2. Inherence (propriety),
3. Clarity,
4. Consistency,
5. Orthogonality (modularity), and
6. Generality.

From the previous discussion, a control case view contributes to the overall completeness of the system architecture by capturing the non-functional aspect of architecture. This also addresses the essential aspect of the real world environment in which the system operates – operating condition, which passes the inherence test. Clarity is given as the concept and the associated rules are easy to understand and can be applied in the architecture modeling. The control case concept provides consistency as it offers non-ambiguity, at the same time it does not conflict with the existing concepts, such as UML. The view also provides orthogonality by separating the non-behavioral requirements from the behavioral nature of use case view. Finally, this passes the generality test as it addresses the non-functional concerns that exist in general application domains.

Therefore, from an architecture modeling perspective, it seems appropriate to extend the 4+1 view to a more complete and real world system model as a 4+2 view.

5. Discussion and Conclusions

The control case view is especially relevant in today's complex IT environment as a result of the constant changing dynamic of IT outsourcing and insourcing. Service level agreements become one of the most vital agreements to system owners as well as the IT service provider. Through the SLA, a business transfers certain IT operational risk to IT service provider; whereas by taking the risk, IT service provider also obtains financial return for providing service guarantees. Therefore an effective and practical SLA brings mutual benefits to both the business system owner and IT service provider. On the other hand, a poorly defined SLA can undermine the relationship between the business and the IT service provider. The control case view can serve as a non-biased view for both parties to appreciate SLRs and subsequently to negotiate and agree on SLAs.

In the UML User Guide, Booch notes that 'the use case view of a system encompasses the use cases that describe the behaviour of the system as seen by its end users, analysts, and testers' [29]. Similarly, the control case view of a system encompasses the control cases that describe the quality of the system as expected by its various stakeholders, i.e. system owners, service providers, operations and users. Moreover, the control case places significance upon the NFR's, which is often the source of SLA contention between the system owner and IT service provider.

The control case approach deals with operational NFRs. For non-operational such as maintainability, and reusability that are not normally associated with a particular operating condition, the control case can be used to document the process, policy or practical guidelines that help the project achieve the softgoal.

The main benefits of the control case approach are summarised below.

- Supports the capture of NFRs during functional requirements gathering by association with its functional use case counterpart.
- Allow the business to understand the risk, impact of the IT system and specify the service level requirements that reflects the trade-off between the cost to implement and the risk to the business.
- Provide a basis for the business and IT service provider to negotiate and reach a service level agreement.
- As a complement to the use case, it allows the project team to fully capture the requirements, better plan the project in terms of task breakdown, timeframe, resource allocation and cost, hence mitigate project failure.

In our extension the control case is associated distinctively with a use case, under its specific operating condition. This provides a clear definition of why and how the NFR is a constraint or system quality associated with a functional requirement.

Finally we observe that further work entails validation of the efficacy of the control case. Such a validation exercise would be most effectively carried out as part of a formal IT project engagement.

Acknowledgements

We thank the anonymous reviewers for the insightful feedback and suggestions to this paper.

6. References

[1] L. Chung, B.A. Nixon, E. Yu and J, Mylopoulos, *Non-Functional Requirements in Software Engineering*, Kluwer Academic Publishers, Boston Hardbound, October 1999.

[2] P.C. Clements and L.M. Northrup, *Software Architecture: An Executive Overview*, Technical Report No. CMU/SEI-96-TR-003, Carnegie Mellon University, Pittsburgh, PA, February, 1996.

[3] C.U. Smith and L.G. Williams, *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*, 1st Edition, Addison-Wesley, September 2001.

[4] P. Kruchten, "Architectural Blueprints ─ The '4+1' View Model of Software Architecture", *IEEE Software*, 12(6), November 1995, pp. 42 -50.

[5] I. Jacobson, "Object Oriented Development in an Industrial Environment", *Conference on Object Oriented Programming Systems and Applications* (OOPSLA '87): Orlando, Florida, October 1987, pp. 183-191.

[6] I. Jacobson, "Use cases - Yesterday, today, and tomorrow", *Software and System Modeling*, 3(3), 2004, pp. 210-220.

[7] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modelling Language Reference Manual*, 2nd Edition, Addison-Wesley, 2005.

[8] P. Kruchten, *The Rational Unified Process: An Introduction*, Addison-Wesley, 1999.

[9] A. M. Davis, and D.A. Leffingwell, "Using Requirements Management to Speed Delivery of Higher Quality Applications", *Rational Software*, 1995.

[10] K. Saleh and A. Al-Zarouni, "Capturing Non-Functional Software Requirements Using the User Requirements Notation", *The 2004 International Research Conference on Innovations in Information Technology* (IIT2004), Dubai UAE, October 2004.

[11] L. Xu, H. Ziv, D. Richardson, and Z. Liu. "Towards Modeling Non-Functional Requirements in Software Architecture", *Proceedings of Early Aspects 2005: Aspect-Oriented Requirements Engineering and Architecture Design Workshop*, Chicago, Illinois, USA, March 2005.

[12] F. Armour and G. Miller, *Advanced Use Case Modelling*, Addison-Wesley, 2001.

[13] L. Laibinis and E. Troubitsyna, "Fault Tolerance in Use-Case Modelling". *Proceedings fourth International*

Workshop on Requirements for High Assurance Systems (RHAS'05), Paris, France, August 29 - Sep 2, 2005.
[14] Q. Tran and L. Chung, "Tool Support for Dealing with Non-Functional Requirements", *Proceedings of IEEE ASSET '99*, Dallas Texas, March 1999.
[15] J. McDermott and C. Fox, "Using Abuse Case Models for Security Requirements Analysis", *Proceedings of 15th Annual Computer Security Applications Conference* (ACSAC '99), Scottsdale USA, 1999, p. 55.
[16] G. Sindre and A. Opdahl, "Eliciting Security Requirements by Misuse Cases", *Proceedings of the 37th Technology of Object-Oriented Languages and Systems* (TOOLS–37 Pacific 2000), Sydney Australia, November 2000, pp. 120–131.
[17] I. Alexander, "Misuse Cases Help to Elicit Nonfunctional Requirements", *Proceedings of 8th International Workshop on Requirements Engineering: Foundation for Software Quality* (REFSQ'02), Essen Germany, September 2002.
[18] P. Loucopulos and V. Karakostas, *Systems Requirements Engineering*, McGraw-Hill, 1995.
[19] L. Chung, J. Mylopoulos, and B. Nixon, "Representing and using non-functional requirements — A process-oriented approach", *IEEE Transactions on Software Engineering*, 8(6), 1992, pp. 483–497.
[20] J. Dobson, "A methodology for analysing human computer-related issues in secure systems", *International Conference on Computer Security and Integrity in our Changing World*, Espoo Finland, 1991, pp. 151–170.
[21] G. Kotonya and I. Sommerville, "A framework for integrating functional and non-functional requirements", *International Workshop on Systems Engineering for Real Time Applications*, Cirencester UK, 1993, pp. 148–153.
[22] I. Jacobson, *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, Boston, 1992.
[23] G. Stoneburner, A. Goguen and A. Feringa, *Risk Management Guide for Information Technology Systems*, National Institute of Standards and Technology (NIST), U.S. Department of Commerce, Publication 800-300, July 2002.
[24] OMG, *UML Profile for Schedulability, Performance and Time*, Object Management Group, 2002.
[25] M. Fowler, *UML Distilled*, 3rd Edition, Addison-Wesley, 2004.
[26] J. Warmer and A. Kleppe, *The Object Constraint Language: Precise Modelling with UML*, Addison-Wesley, 1999.
[27] R. Youngs, D. Redmond-Pyle, P. Spaas, and E. Kahan, "A Standard for Architecture Description", *IBM Systems Journal*. Vol 38, No 1. 1999.
[28] W.B. Teeuw, H. Van den Berg, "On the Quality of Conceptual Models", *Proceedings of 16th International Conference on Conceptual Modeling* (ER'97), Los Angeles USA, November 1997.
[29] G. Booch, I. Jacobson, and J. Rumbaugh, *UML User Guide*, Addison-Wesley. 1998.
[30] G. Schneider and J.P. Winters, *Applying Use Cases 2nd Edition A Practical Guide*, Addison-Wesley, 2001.
[31] S. Adolph, P. Bramble, A. Cockburn, A. Pols, *Patterns for Effective Use Cases*, 1st Edition, Addison-Wesley, 2002.
[32] A. Cockburn, Basic Use Case Template, Humans and Technology, Technical Report TR.96.03a, October 1998.

Appendix I. Detailed Control Case Example

Detailed example of a control case is shown below and is completed with arbitrary conditions and constraints to illustrate usage.

CONTROL CASE: Control response time		
Control Case ID: CC-001		
Operating condition: Concurrent users load.		
Description: Under multiple concurrent users load, the system's response time may become unacceptable to end users which lead to the risk of losing customers for the business. This control case defines the maximum response time for various transactions that the system must meet in order to mitigate the risk.		
NFR Category: Performance and Capacity.		
Associated Use Cases: Register user, Authenticate User, Transfer fund, Check Balance, Pay Anyone.		
Regulatory Constraints: Audit trail for each transaction must be captured, stored and retrievable.		
Business Constraints: Payment transaction needs one working day for clearance.		
Technical Constraints: Multi-tier J2EE architecture.		
Vulnerability: Uncertain bandwidth in internet environment, dependency on ISP network infrastructure.		
Threat Source: Hackers, crime syndicates, unexpected user load.		
Operating conditions	**Control Target (SLR)**	
1. Peak load time: 10am-12am, 2pm-4pm week day, 10am-12pm Saturday Maximum concurrent active sessions <= 10,000. **2.** Normal Load time: outside peak load time, Maximum concurrent active sessions <= 1,000.	**1.** 90% of the user response time should be less or equal than 5 seconds. 95% of the user response time should be less than 10 seconds. **2.** 90% of the user response time should be less or equal than 3 seconds. 95% of the user response time should be less than 8 seconds.	
Impact if SLR not met: Estimate that the business will lose 20% customers to competitors.		
Impact Rating	**Likelihood**	**Risk Rating**
High	High	High
Controls: Load balancing and clustering of the application. Use cache to improve response time. Negotiate network bandwidth SLA with ISP.		
Residue Risk: Low		