Issues for Evaluating Reliability in Software Architectures

Grimán, A.
Processes and Systems Department – LISI
Universidad Simón Bolívar
Caracas – Venezuela
agriman@usb.ve

Valdosera, L.
Processes and Systems Department – LISI
Universidad Simón Bolívar
Caracas – Venezuela
vvalldosera@tspven.com

Mendoza, L.
Processes and Systems Department – LISI
Universidad Simón Bolívar
Caracas – Venezuela
lmendoza@usb.ve

Pérez, M.
Processes and Systems Department – LISI
Universidad Simón Bolívar
Caracas – Venezuela
movalles@usb.ve

Méndez, E.
Processes and Systems Department – LISI
Universidad Simón Bolívar
Caracas – Venezuela
emendez@usb.ve

ABSTRACT

Currently, the requirements of Business sector promote more and more complex Information Systems. Reliability is one of the quality characteristics widely expected by users and developers. This characteristic is architectural by nature since it can be directly promoted by software architecture. This relation determines the importance of designing architectures that guarantee reliable systems.

This article presents a research in progress whose objective is developing an architectural evaluation method based on Reliability. The first step considered for designing the method included: the construction of a Conceptual Model, a model to specify the architectural quality based on Reliability (Utility Tree), a set of scenarios associated to this characteristic. The first model allows identifying the concepts inherent to Reliability and their relationships; the second one covers all quality features related to Reliability in order to specify it; and the scenarios guide the software architect for anticipating context stimulus and evaluating the architectural responses.

Keywords
Quality Evaluation, Reliability, Software Architecture.

INTRODUCTION

Kazman et al. (2000) state that if a Software Architecture is a key asset for an organization, then architectural analysis should be a key practice for this organization. It is essential for organizations to count all the time on reliable systems, since their operations and corporate data protection depend upon them. Reliability is one of the quality attributes that should be offered by the systems to their users and, since quality characteristics of the software system are determined mainly by their Architecture (Bass et al., 2003), it is fundamental to adopt strategies to assure reliability of Architecture and, thus, of the System. The different architectural strategies guaranteeing reliability include components reuse and the use of such architectural and design patterns and architectural styles in the software design that guarantee Reliability.

In this regard, this research in progress is aimed at developing a Method for the Reliability Assessment of Software Architectures. To this end, the role of Reliability in Software Quality is first described; a Conceptual Model for Reliability in Software Architectures is established, followed by a Utility Tree as a technique to specify Architectural Quality. Then the potential Reliability Scenarios, which should be considered for the Architecture when developing an Information System, are presented to close with conclusions and recommendations.

RELIABILITY CONCEPTUAL MODEL IN SOFTWARE ARCHITECTURE

To specify the issues related to Evaluating Reliability in Software Architectures, an Ontology Creation Methodology was employed: Ontology Development 101 (Noy and McGuinnes, 2001).

With the boom of Web-based distributed systems, Reliability has become a very desirable quality characteristic. Subjects or concepts related to it abound in recent literature; however, this relationship is not mature. Consequently it was necessary to create a model to represent concepts and their relationships with Reliability. The idea is to provide researchers and students in this area with a reference framework.

Figure 1 show a set of concepts related to the Reliability of a Software Architecture. It can be observed that there is a large number of conceptual relationships that, when considered, shall help to perform a much more systemic assessment of the architecture, which will translate in a much more objective and effective selection of the software architecture, ideal for the development of Information Systems (IS). Some of the concepts shown in these figures are described below.

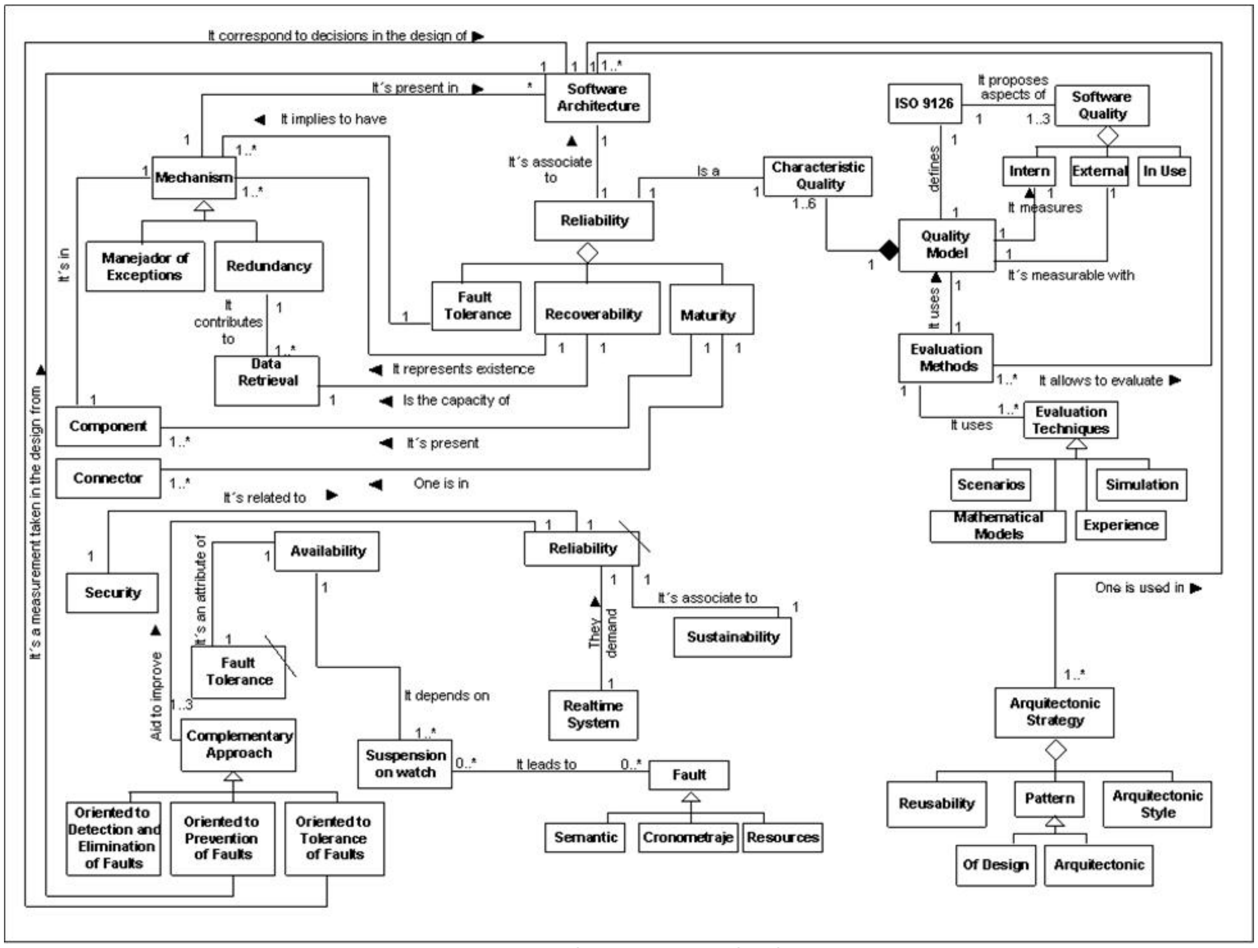

Figure 1. Conceptual Model for Reliability of Software Architecture.

ISO/IEC 9126 (2000) proposes three quality issues: Internal Quality, measurable from the intrinsic characteristics; External Quality, measurable based on product behavior; and Use Quality, measurable during the effective use by the user. ISO/IEC 9126 (2000) defines a Quality Model to measure Internal and External Quality of a product.

This Quality Model consists of six (6) Quality Characteristics to be considered at the moment of developing a Quality software. ISO 9126 (2000) defines Reliability as one of these Quality characteristics. Reliability sub-characteristics are fault tolerance, maturity and recoverability (ISO/IEC 9126, 2000).

Barbacci et al. (1997) classify system faults into: timing faults (timing overflows), semantics faults (wrong output values), and resources faults (memory overflow due to misapplied pointers). Sommerville (2000) stresses that there are three complementary approaches used to improve system Reliability: fault prevention, detection and removal; and fault tolerance.

Pressman (2003) states that all software faults are the result of design or implementation problems. Therefore, decisions regarding techniques for fault prevention and tolerance are made during the Architecture design, whereas decisions as to techniques for fault *detection* and *removal* are made in the system testing phase. Sommerville (2000) points out that these three complementary approaches help improve system Reliability.

Bishop (2003), Bass et al. (1997), and Sommerville (2000) confirm that Availability is closely related to Reliability, although it is not directly specified in ISO 9126 Standard. Losavio et al. (2003) state that Availability is an attribute of fault tolerance. Availability directly depends on service suspensions in a software system.

Rogina (1999) states that Real-Time Systems should be reliable, since they have to be functioning although faults occur. Service suspension in these systems leads to critical or catastrophic situations, resulting Reliability a key characteristic of Real-Time Systems. Reliability measurements are focused on time proportions predicting system faults that lead to service suspensions.

Bishop (2003) states that Security is closely related to Reliability, since access in a system should be granted only to the allowed actors, and if the actor is not the one authenticated by the system, certain states guaranteeing system integrity should be assumed by default.

Losavio et al. (2003) assure that sub-characteristic Maturity is present in the components and connectors of the software system. Losavio et al. (2003) stress that fault tolerance implies having mechanisms or software devices guaranteeing the software existence.

A mechanism can be in a component or integrated into a software *component* (Losavio et al., 2003). A mechanism can be exceptions manager or redundancy manager (Losavio et al., 2003). Fault tolerance exclusively depends on the system Architecture.

Recoverability represents the existence of *mechanisms* or software devices (Losavio et al., 2003). *Redundancy* favor data recovery (Losavio et al., 2003). Recoverability is the data recovery capability in a software system (ISO/IEC 9126, 2000). Kazman et al. (1998) state that Software Architecture determines or mainly favors the system Quality characteristics, Reliability among them.

It is necessary to apply strategies at the architectural levels to ensure Software Reliability. The different architectural strategies guaranteeing Reliability are components reuse and the use in software design of architectural patterns, design patterns, and architectural styles favoring Reliability.

Architectural decisions are used in Software Architecture. Kazman et al. (2000) point out that one of the main reasons motivating architectural Evaluation is that Software Architectures promote the characteristics of System Quality. All approaches toward the achievement of system Quality characteristics are of architectural nature.

Evaluation methods determine the Architecture capability to support Quality characteristics. Evaluation methods are based on a Quality Model that specifies the Quality characteristics to be evaluated.

Architectural Evaluation Methods require the use of evaluation techniques. The different kinds of techniques are: Scenarios, Mathematical Models, Experience, and Simulation (Grimán et al., 2002).

The method that will be designed for the Reliability Assessment of Software Architecture (MECAS) is based on the Quality Model provided by the ISO/IEC 9126 international standard. Non-functional or quality requirements are refined to sub-characteristics that can be measured at the architectural level according to ISO/IEC 9126, and that are addressed through a Utility Tree that will serve as a tool to specify Architectural Quality.

SPECIFICATION OF ARCHITECTURAL QUALITY – UTILITY TREE

A Utility Tree (UT) is a technique to transfer the goals of the quality characteristics of the system to Quality Scenarios that can be proven. UA also helps to elicit a definition of the Quality requirements of the system in a practical and operational sense that can be understood by the stakeholders (Jones and Lattanze, 2001).

In this sense, Figure 2 shows an example of a UT for Reliability and the associated Quality attributes than can be generated to assess Reliability in Software Architectures and should be instantiated while MECAS.

In this UT, Reliability is established as the utility (root). Then Quality characteristics such as Maturity, Fault tolerance, and Availability are determined and attributes associated to each characteristic are refined by describing Reliability scenarios, prioritized according to risk and/or scope. A quality attribute can be represented by one or more scenarios. For example, six (6) attributes would exist in Availability, represented by the six (6) scenarios established for this characteristic.

The notion of assessment of Quality characteristics, based on context, leads to adopt Scenarios as a descriptive means for specifying and assessing Quality characteristics in Software Architecture (Kazman, 1999).

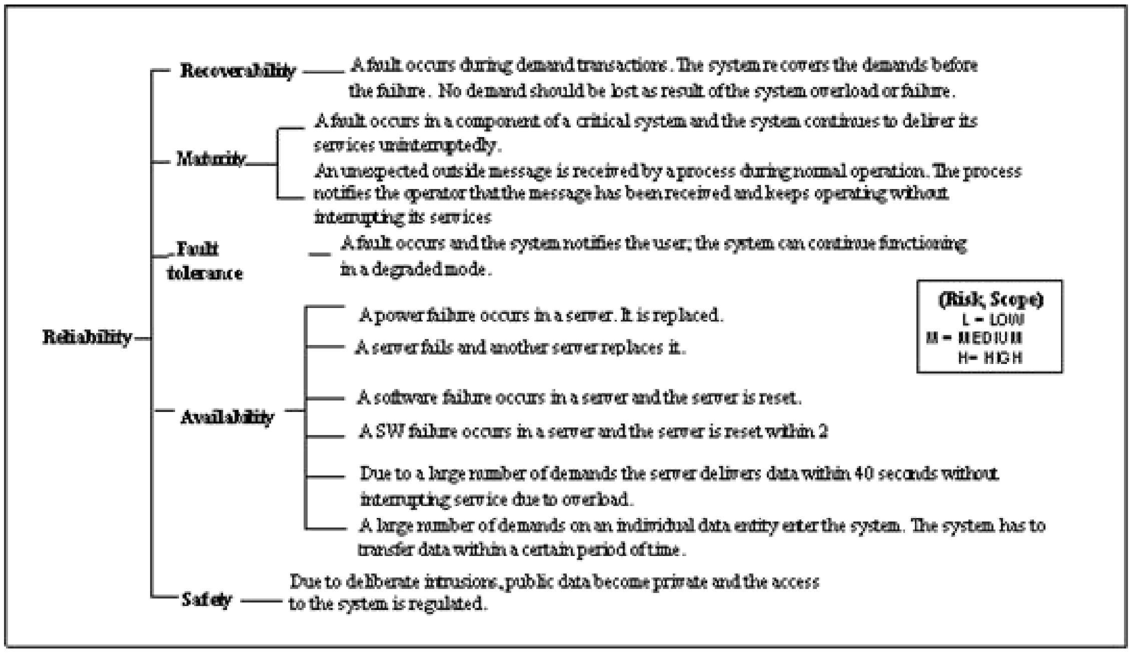
Reliability
Recoverability
A fault occurs during demand transactions. The system recovers the demands before the failure. No demand should be lost as result of the system overload or failure.
Maturity
A fault occurs in a component of a critical system and the system continues to deliver its services uninterruptedly.
An unexpected outside message is received by a process during normal operation. The process notifies the operator that the message has been received and keeps operating without interrupting its services
Fault tolerance
A fault occurs and the system notifies the user; the system can continue functioning in a degraded mode.
Availability
A power failure occurs in a server. It is replaced.
A server fails and another server replaces it.
A software failure occurs in a server and the server is reset.
A SW failure occurs in a server and the server is reset within 2
Due to a large number of demands the server delivers data within 40 seconds without interrupting service due to overload.
A large number of demands on an individual data entity enter the system. The system has to transfer data within a certain period of time.
Safety
Due to deliberate intrusions, public data become private and the access to the system is regulated.
(Risk, Scope)
L = LOW
M = MEDIUM
H= HIGH

Figure 2. Example of a Utility Tree for Reliability.

RELIABILITY SCENARIOS

A scenario can be conceived as a framework conducting operations in the other structures. Scenarios, besides clarifying requirements, help to prioritize which parts of the architecture should be elicited in the first place. A quality characteristic can be used to motivate the creation of a scenario; but the impact of the scenario on other Quality characteristics should also be considered (Kazman, 1999).

The Reliability analysis starts by considering several fault scenarios. The goal of the Architecture is to manage the different types of faults: timing, semantic and system faults. The description of the fourteen (14) Reliability Scenarios proposed in this research (see Table 1) consists of the six elements defined by Kazman (1999): stimulus, its source, response, environment, stimulated artifact, and response measurement.

SCENARIO
Scenario 1: A system suffers a software failure in normal operation and is reset. *Stimulus: A software failure occurs.* *Source of stimulus:* Internal. *Response:* Reset. *Environment:* Normal operation. *Stimulated artifact:* System. *Response measurement:* Reset time.
Scenario 2: A power failure occurs in a system in normal operation and it is replaced.
Scenario 3: One of the servers fails in normal operation. Another server assumes operation.
Scenario 4: A failure occurs and the system notifies the user; the system can continue functioning in degraded mode.
Scenario 5: The demands for electronic FTP come to a site where the FTP server is low, the system is suspended for a period of time from the first failed demand and all resources are available while demands are suspended.
Scenario 6: A failure occurs and the system can interrupt its service for a determined period of time. This interruption is not measured versus the system availability unless it exceeds a well-defined interval.
Scenario 7: Demands for the electronic FTP come to a site where the FTP server is low; the system is suspended for a period of time from the first failed demand. The user with the failed site continues sending new orders every 10 minutes, the system queues the demands.
Scenario 8: A failure occurs during demand transaction in normal operation. The system recovers the demands before the failure. No demand should be lost as a result of the overload or failure of the system.
Scenario 9: Due to previous deliberate intrusions into the system, public data are transformed into private data and access is regulated in normal operation.
Scenario 10: In Normal operation, A failure occurs in a component of a critical system and it continues providing its services uninterruptedly.
Scenario 11: A mistake in the replication process results in a loss of synchronization of a transaction in the database with the backup of the database. The transaction is synchronized with the backup.
Scenario 12: A large number of customers need access to the server side object. The server has to deliver data within a determined response time.
Scenario 13: A large number of demands on an individual data entity come to the system from a user interface under normal conditions. The system has to transfer data within a determined period of time.
Scenario 14: An unexpected external message is received by a process during normal operation. The process informs the operator that the message has been received and continues operating without interrupting its services.

Table 1. Scenarios proposed.

The Conceptual Model, Utility Tree, and Reliability Scenarios presented in the previous sections will serve as input for the Design of a Method for Reliability Assessment of Software Architectures, which will provide guidance for stakeholders involved in the development process of IS when there is necessary to decide which is the most reliable software architecture to achieve a quality IS.

CONCLUSIONS AND RECOMMENDATIONS

These preliminary results are intended to design a Method for Reliability Assessment of Software Architecture oriented toward comparison of two potential Architectures with the view to selecting the one that best satisfies the initial quality requirements of the system and that will be further improved during the Transformation of the Architecture. MECAS will be inspired in the ATAM method (Architecture Trade-off Analysis Method) but customized for Reliability; it will have to be refined and tested in an organization devoted to Information Systems development with the aim of determining its effectiveness. To do this, the DESMET Methodology will be use.

It is established that Reliability is not a trivial quality characteristic but it relates multiple variables that have to be measured and to which the system architecture should response.

REFERENCES

1. Barbacci, M., Kein, M. and Weinstock Ch. (1997) Principles for Evaluating the Quality Attributes of a Software Architecture, Recuperado el 03 de Agosto de 2004 en www.sei.cmu.edu/pub/documents/96.reports/pdf/96tr036.pdf
2. Bass, L., Clements, P., Kazman, R. and Bass K. (1997) Software Architectures, Addison-Wesley Pub Co.
3. Bass, L., Clements, P. and Kazman, R. (2003) Software Architecture in Practice, Segunda Edición, Addison-Wesley.
4. Bishop, M. (2003) Computer Security - Art and Science, Addison-Wesley
5. Grimán, A., Pérez, M. and Mendoza, L. (2002) Evaluación Arquitectónica de Software basada en la Técnica de Escenarios, AsoVAC 2002
   Barquisimeto, Venezuela. 17-22 de noviembre de 2002, Recuperado el 18 de Junio de 2004 en http://www.lisi.usb.ve/publicaciones/Evaluación%20Arquitectónica.pdf
6. ISO/IEC 9126 (2000) ISO9126 International Organization for Standarization Software Engineering Product Quality.
7. Jones, L. and Lattanze, A. (2001) Using the Architecture Tradeoff Analysis Method to Evaluate a Wargame Simulation System: A Case Study, Recuperado el 22 de Agosto de 2004 en http://www.sei.cmu.edu/pub/documents/01.reports/pdf/01tn022.pdf
8. Kazman, R., Klein, M., Barbacci, M., Longstaff T., Lipson H. and Carriere J. (1998)The Architecture Tradeoff Analysis Method, Recuperado el 08 de Junio de 2004 en http://www.sei.cmu.edu/pub/documents/98.reports/pdf/98tr008.pdf
9. Kazman, R. (1999) Using Scenarios in Architecture Evaluations, Recuperado el 03 de Agosto de 2004 en http://www.sei.cmu.edu/news-at-sei/columns/the_architect/1999/June/Architect.jun99.pdf
10. Kazman, R., Klein, M. and Clements, P. (2000) ATAM: Method for Architecture Evaluation, Technical Report CMU/SEI-2000-TR-004, Recuperado el 05 de Junio de 2004 en http://www.sei.cmu.edu/pub/documents/00.reports/pdf/00tr004.pdf
11. Losavio, F., Chirinos, L., Lévy, N. and Ramdane-Cherif, A. (2003) Quality Characteristics for Software Architecture, Journal of Object Technology, 2, 2, pp. 133-150, Recuperado el 20 de mayo de 2004 en http://www.jot.fm/issues/issue_2003_03/article2.pdf
12. Noy, N. F. and McGuinness, D. L. (2001) Ontology Development 101: A Guide to Creating Your First Ontology, Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880.
13. Pressman, R. (2003) Ingeniería del Software. Un enfoque práctico, McGraw-Hill.
14. Regina, P. (1999) Tolerancia a Fallas en Sistemas de Tiempo Real, Recuperado el 13 de Julio de 2004 en www.dc.uba.ar/people/proyinv/cso/rt- minix/Tesis.doc+sistemas+en+tiempo+real.html
15. Sommerville, I. (2000) Ingeniería del Software, Addisson-Wesley.