TOWARDS A MAINTAINABILITY EVALUATION IN SOFTWARE ARCHITECTURES

Anna Grimán, Luisana Chávez, María Pérez, Luis Mendoza, Kenyer Domínguez
Processes and Systems Department – LISI
Universidad Simón Bolívar
Caracas – Venezuela

Keywords: Architectural Evaluation, Maintainability, Software Architecture, Architectural mechanism, Scenario.

Abstract: Maintainability is an internal quality characteristic that is contemplated by many users and developers, and therefore is deeply related to software architecture. It presents an organization of its components and relation which promote or obstruct different attributes like testability, changeability, and analyzability. As part of a research in progress, this article analyzes and organizes a set of architectural mechanisms that guarantee software maintainability. To propose the architectural mechanisms we decided first to construct an ontology, which helps identify all concepts related to Maintainability and their relationships. Then we decided to focus and specify mechanisms that promote maintainability, also we present a set of scenarios that will explore the presence at the architecture of those concepts previously identified, including the architectural mechanism analyzed. With the products described in this article we have the bases to develop an architectural evaluation method, which is based on maintainability.

1 INTRODUCTION

Bass et al. (2003) state that Architecture defines the most fundamental design decision and largely permits or precludes a system's quality attributes such as performance or maintainability. Over the last few years, fulfilling quality requirements of the system has become more important than providing functionality requirement. Maintainability is one of the quality characteristics that systems should have, in order for a software product to change and evolve. New user requirements are often appearing after the first delivery, maintenance is therefore required. Frameworks, architectural styles, architectural and design patterns are some of these mechanisms that help assure Maintainability.

In this context, our research, which is now in progress, is an attempt to develop an architectural evaluation method, which is based on maintainability. To achieve this objective, an ontology for Maintainability evaluation in software architectures is established, followed by the description of mechanisms that ensure Software Maintainability in software architectures, afterwards a description of maintainability scenarios will be made to close with conclusions and recommendations.

Also, this article analyzes and organizes a set of architectural mechanisms that guarantee software maintainability.

2 MAINTAINABILITY ONTOLOGY IN SOFTWARE ARCHITECTURE EVALUATION

We have created a model to represent concepts and their relationship with Maintainability, because in other recent literature these relationships are not mature enough. Figure 1 shows a set of concepts related to the Maintainability Evaluation in Software Architectures. 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 Figure 1 are briefly described below.

Grimán A., Chávez L., Pérez M., Mendoza L. and Domínguez K. (2006).
TOWARDS A MAINTAINABILITY EVALUATION IN SOFTWARE ARCHITECTURES.
In *Proceedings of the Eighth International Conference on Enterprise Information Systems - ISAS*, pages 555-558
DOI: 10.5220/0002462005550558

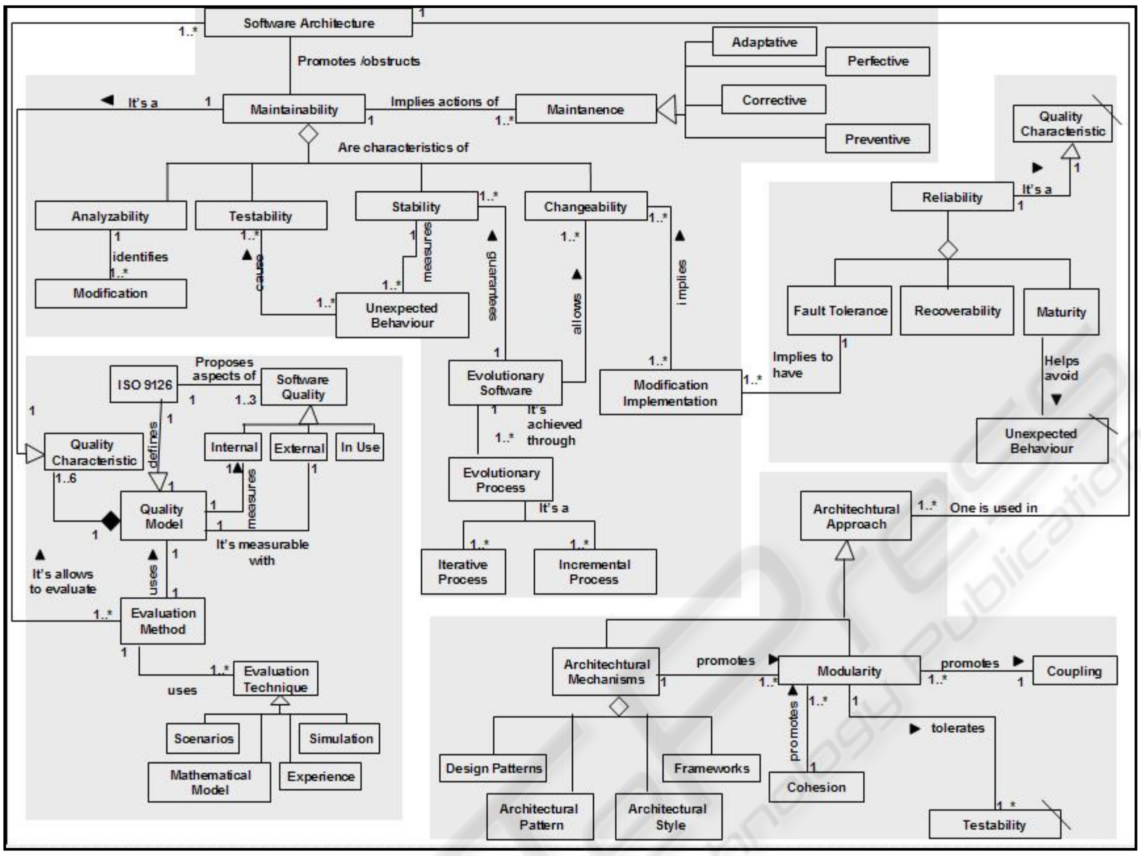

Figure 1: Ontology for Maintainability Evaluation in Software Architectures.

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. This model should be based on ISO 9126 standard. Furthermore, a variety of qualitative and quantitative techniques are used for analyzing specific quality attributes, Barbacci (1995). Bosch (2000) states that architectural evaluation methods require the use of evaluation techniques. The different kinds of techniques are: Scenarios, Mathematical Models, Experience, and Simulation.

ISO 9126-1 proposes three aspects by which software quality of a product can be measured: External quality, measurable based on how the software product acts or responds; internal quality, measurable from the intrinsic characteristics of the software product; and Use Quality, measurable from the correct usage given by the user. These aspects inspire the ISO 9126-1 quality model (ISO/IEC, 2002). ISO 9126-1 quality model, proposes a set of six independent high-level quality characteristics, which are defined as a set of attributes of a software product by which its quality is described and evaluated. Maintainability is one of these six characteristics defined by ISO 9126 (2002). Maintainability sub-characteristics are stability, changeability, analyzability, and testability (ISO.IEC 9126, 2002). This characteristic is related to software evolution and Maintenance process.

ISO/IEC 14764 defines four categories of maintenance, as follows: Corrective, Adaptive, Perfective, and Preventive maintenance. These four kinds of maintenance imply different needs. Maintainability then relates to the easiness to perform maintenance activities. It also relates with software evolution during development and after software delivery.

Larman (2003) points out that evolutionary software is a software development process by which the software product is delivered in various versions. This process however, does not exclude the management of changes during the development stage. Successive versions may provide a complete implementation of the actual specification; however, it provides a changed implementation correspondent to the changes made on the product specification, both the additional requirements as the removed requirements.

Table 1: Benefits of Mechanisms that cause software Maintainability.

Mechanism		ISO 9126 -1	Mechanism	ISO 9126 -1
Design Pattern	Builder	Analyzability, Changeability, Testability.(Modularity),	Memento	Stability, Testability, Changeability.
	Adaptor	Changeability. (Reuse of Code).	Bridge	Changeability, Analyzability, Stability
	Iterator	Changeability (Coupling)	Prototype	Analyzability, Stability, Changeability
	Chain of Responsibility	Testability, Changeability		
Architectural Pattern	Pipes and Filters	Changeability (Coupling, Modularity).	Reactor	Testability (Modularity)
	Interceptor	Analyzability, Changeability	Proxy	Changeability, Analyzability. (Coupling)
Architectural Style	Layer System	Analyzability, Testability. (Code Reuse) Changeability	Object Oriented	Testability, Changeability. (Modularity), Analyzability
	Component Based	Testability, Changeability (Modularity)		
Framework		Testability, Changeability (Modularity)		

Evolutionary Software is achieved through an incremental and iterative process. Evolutionary software guarantees software stability, because it helps avoid unexpected behaviors.

Mechanisms and other architectonic decision (such as effective modularity) should be applied at certain architectural levels in order to assure Maintainability. The different architectural approaches that guarantee software maintainability are architectural patterns, design patterns, architectural styles, and frameworks. Each of these will be studied and analyzed in the next section as a translation to ISO 9126-1.

3 ARCHITECTURAL MECHANISMS

Table 1 shows the architectural mechanisms studied that promote maintainability of a Software Architecture. These mechanisms can be linked to maintainability sub-characteristics defined by ISO 9126-1, which will be described bellow.

Buschman et al. (1996) establishes that design patterns supply a diagram that refines subsystems, components of a software system or the relationship between them. Some of the design patterns that Gamma et al. (1994) propose may cause software maintainability. The ones we believe cause this quality characteristic are shown in Table 1.

Shaw et al. (1996) define an architectural style as a description of component types and their topology, which includes a description of the pattern of data and control interaction among the components. Architectural Styles that cause maintainability are shown in Table 1.

On the other hand, Szyperski et al. (2002) defines framework as a set of cooperating classes, some of which may be abstract, that make up a reusable design for a specific class of software. Frameworks are not necessarily domain specific, however they are concept specific. We can then conclude that frameworks enhance modularity and therefore maintainability by covering implementation details with simple and stable interfaces. Frameworks modularity may help locate changes, reducing the effort to maintain the software.

Table 1, shows how each mechanism can be interpreted as one of the sub-characteristics of Maintainability defined by ISO 9126-1.

4 MAINTAINABILITY SCENARIOS

Scenarios have been widely used and documented as a technique during requirements elicitation, specially with respect to the operator of the system according to Bass et al. (2003). They have also been widely used during designs a method of comparing design alternatives.

The Maintainability analysis starts by considering scenarios of change to the software product. The goal of the architecture is to control in a subtle way all the different types of maintenance:

corrective, adaptive, perfective, and preventive. Twenty (20) Maintainability Scenarios were proposed in this research. These are all exploratory scenarios that seek a way to determine the presence of some of the concepts identified during the elaboration of the ontology. A set of them are described bellow.

Scenarios for Corrective Maintenance:

1. An attribute must be added to the constructor of a class in order to correct a fault.
2. A link of the home page of a web application must be erased in order to reduce confusion of users.

Scenarios for Perfective Maintenance:

3. A method must be added to a class in order to add functionality.
4. A new class must be created and it should inherit all properties from another class.

Scenarios for Adaptive Maintenance:

5. A two layer platform must be migrated to a three layer one; the new layer must be a web Interface.
6. A class is needed, and its interface doesn't match the one that is needed.

Scenarios for Preventive Maintenance:

7. An external audit must be made to verify functionality.
8. An external audit must be made to verify effectiveness.

To build this table, the scenario is broken down into its Stimulus and Response, in order to ensure that each one has been captured accurately. Each scenario generates a sequence of steps. These steps provide support for the group discussion, which leads to the Architectural Decisions, the Risks, Non-risks, Sensitivity Points and associated Tradeoffs. An example of how each of the scenarios is broken down is shown in Table 2.

Table 2: Analysis of Scenario #17.

Scenario #17 An external audit must be made to verify functionality.
Attribute: Maintainability
Environment: During preventive maintenance work
Stimulus: External Audit in response to functionality verification.
Response: -Existence of a mechanism, module or component that support and registers transactions. - Existence of a mechanism, module or component that stores modification history of data structures and architecture. - Existence of a mechanism, module or component that allows backup and restores data and configurations. - Existence of a mechanism, module or component that allows remote administration.

The Ontology, the Architectural Mechanisms, and Maintainability Scenarios presented in the previous sections will serve as input for the Design of a Method for Maintainability Assessment of Software Architectures.

5 CONCLUSIONS AND FUTURE RESEARCH

As established, Maintainability is a very important quality characteristic but it relates to multiple issues that should be taken in to consideration and measured. A system Architecture should respond to these variables, therefore the study of software maintainability is very complex. It is not a characteristic that can be studied apart from reliability, and all its different types (perfective, corrective, preventive, and adaptative) should always be considered. An evaluation method should include scenarios in combination with other techniques. These scenarios help understand architectural aspects that are not easy to determine. The final goal is to implement an evaluation method that makes the evaluation process more efficient.

REFERENCES

Bass,L., Clements,P.,& Kazman,R.(2003). *Software Architecture in Practice, 2nd edition*. Addison Wesley.

Buschman F., Meunier R., Rohnert H., Sommerlad P., & Stal, M.(1996). *Pattern-Oriented Software Architecture.* New York. John Wiley & Sons Inc.

Shaw M., & Garlan D. (1996). *Software Architecture – Perspective of an Emerging Discipline*. Upper Saddle River, New Jersey. Prentice Hall.

Bosch J. (2000). *Design and Use of Software Architecture.* Harlow, England .ACM Press.

Szyperski,C.(2002). *Component software: Beyond Object-Oriented programming*. Addison-Wesley.

Barbacci,M.,Klein,M.H.,Longstaff,T.A., & Weinstock, C.B. (1995). *Quality Attributes*. Technical Report, CMU/SEI-95-TR-021, December 1995.

ISO/IEC 9126-1:2001. (2001) Software Engineering-Product Quality-Part 1: Quality Model, ISO and IEC.

ISO/IEC 14764-1999.(1999). Software Engineering-Software Maintenance, ISO and IEC, 1999.

Larman,C.(2003). Agile and Iterative Development: A Manager's Guide. Addison Wesley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns, Addison-Wesley.