AN ONTOLOGY FOR ARCHITECTURAL EVALUATION
Case Study: Collaboration Systems

Anna Grimán, María Pérez
Processes and Systems Department – LISI
Universidad Simón Bolívar
Caracas – Venezuela

José Garrido, María Rodriguez
Software Engineering Department – LSI
Universidad de Granada
Granada – España

Keywords: Software Quality, Software Architecture, Evaluation, Collaboration Systems.

Abstract: Barbacci et al. (1995) state that the development of systematic ways to relate the quality attributes of a system to its architecture, it constitutes the basis for making objective decisions on design agreements, and helps engineers do reasonably accurate predictions as to the system attributes, free of prejudice and non-trivial assumptions. The aim is being able to evaluate architecture quantitatively to reach agreements among multiple quality attributes and thus globally attain a better system. However, the elements required to incorporate this evaluation into different types of development models, are not clear. This paper proposes an ontology to conceptualize the issues inherent to architectural evaluation within a development process, which will help identify the scope of the evaluation, as well as the issues to be guaranteed to achieve effectiveness within different development processes, both agile and rigorous. The main conclusion of the research allowed us to identify the interaction elements between the development process and an architectural evaluation method, establishing the starting and end points as well as the inputs required for the incorporation into different kinds of processes. This interaction was validated through a case study, a Collaboration Systems Development Methodology.

1 INTRODUCTION

According to Barbacci et al. (1995) software quality is defined as the degree of the desired combination of attributes. These attributes are additional requirements of the system (Clements et al., 2002), different to the functional requirements, which refer to characteristics that the system should have.

Bosch (2000) states that quality requirements are highly influenced by the system architecture. In this regard, Bass et al. (2003) state that system quality should be considered throughout all the design phases, but quality attributes are promoted differently throughout them.

Since architecture is crucial for quality, an architecture analysis can, and should, be performed to evaluate how satisfactory it is for the intended purpose (Bass et al., 2003). However, the evaluation criteria should be fully clear before the architecture analysis is started.

Due to the significance of architectural decisions, they obviously receive particular attention. It is always more cost-effective to evaluate software quality as early as possible in the life cycle (Bass et al., 2003).

Based on this premise, it is necessary to count on methods for the early architectural evaluation of software quality, considering the issues of the development process involved in this evaluation to ensure the inputs required. The goal of his work is to conceptualize the issues inherent to the architectural evaluation into the development process, through an

Grimán A., Pérez M., Garrido J. and Rodriguez M. (2006).
AN ONTOLOGY FOR ARCHITECTURAL EVALUATION - Case Study: Collaboration Systems.
In *Proceedings of the Eighth International Conference on Enterprise Information Systems - ISAS*, pages 310-315
DOI: 10.5220/0002462803100315
Copyright © SciTePress

ontology used to identify the evaluation scope and to ensure its effectiveness within different development processes, both agile as well as rigorous.

The paper presents a review of the theoretical constructors upon which the research is based, and the methodology applied for the creation of ontologies, followed by their description in terms of model and their meanings; then an instantiation of the developed model by applying a case study is presented, and finally conclusions and problems for future research are pointed out.

2 SOFTWARE ARCHITECTURE AND QUALITY ATTRIBUTES

Most authors (Bass et al., 2003; Clements et al., 2002; Hofmeister et al., 2000; Buschmann et al., 1996) consider that software architecture defines the system structure. This structure consists of components – modules or pieces of codes – which arise from the notion of abstraction, perform specific functions, and interact with each other exhibiting a defined behaviour. Shaw and Garlan (1996) state that such structural issues include organizational and global control structures, communication protocols, synchronization, data access, functionality allocation to design elements, physical distribution, composition of elements of design, scalability and performance, and selection among design alternatives..

Bass et al. (2003) point out that to meet a specific attribute it is necessary to make architectural decisions which require a little knowledge of functionality. They also establish that when the architect is considering a decision to software architecture, he or she asks him/herself which will be the impact of the decision on certain attributes. Based on this, they state that every decision incorporated into software architecture can have a potential impact on a set of quality attributes. Therefore, the significance of a software system architecture is recognized as the design basis of a system (Kruchten, 2003), and as an artefact determining quality attributes (Clements et al., 2002).

Until recently there were not general usable methods to evaluate software architecture (Clements et al., 2002). If there was one, its approach would have been incomplete, ad hoc, non repeatable, and little reliable. Accordingly, multiple evaluation methods have been proposed (Bosch, 2000; Clements et al., 2002; In et al., 2001), which use different techniques to evaluate software architecture quality.

3 ONTOLOGY METHODOLOGY

To specify the issues related to a early evaluation of software quality within the development process, an Ontology Creation Methodology was employed as starting point: Ontology Development 101 (Noy and McGuinnes, 2001). In this research, this methodology was selected because of: a) it proposes a semi-formal specification for conceptualization, such as classes and relations between them, and b) it promotes an iterative, top-down approach. These characteristics made this methodology adequate to our research objective.

The steps that Noy and McGuinnes (2001) propose for their methodology are: Determine the domain and scope of the ontology, Consider the reuse of existing ontologies, List relevant terms in the ontology, Define classes and hierarchy of classes, Define classes –properties of slots (classes, hierarchy of classes and properties), Define the slots facets (ontology), Create instances (ontology and the modelled domain).

4 ONTOLOGY FOR ARCHITECTURAL EVALUATION

As suggested by Noy and McGuinnes (2001), an iterative process was followed. It began with the conceptualization of Architectural Evaluation of Software Quality within a Development Process. Then the steps were repeated for those complex concepts requiring special detail (highlighted). For this reason, the final model developed present a set of shared concepts derived from the relationship between main dimensions.

Concepts for the Evaluation of Software Architecture: According to Sommerville (2005) the Design of the system is the stage at which the structure is designed based on the specifications. The software architecture is the key artefact of design discipline. Software architecture can be considered the system structure as a function of components definition and their interactions (Bass et al., 2003), organized in models and views.

Bosch (2000) states that the imposition of determined architectural styles increases or reduces the possibility of satisfying certain system quality attributes. Similarly, he proposes the use of architectural patterns and design patterns to meet the system quality requirements. These concepts (architectural styles and patterns) are not widely differentiated in reviewed literature; nevertheless, Kruchten (2003) establishes that the style may be defined by a set of patterns. According to this author some mechanisms are embedded in architectural styles and patterns.

Software architecture can be then considered as the "bridge" between the system requirements and implementation (Hofmeister et al., 2000). In this sense, Bosch (2000) states that software architecture evaluation is a non-trivial task, since the goal is to measure system properties based on abstract specifications, for example architectural designs. This evaluation should produce results directly observable for the architect.

On the other hand, the additional system characteristics or quality attributes are closely related to the intended use of the proposed system (Clements et al., 2002), because the domain defines the system behaviour, without forgetting functionality (Barbacci et al., 1995). Therefore an analysis of the system context is necessary, because this provides abundant information on its quality. Most quality attributes can be differently organized and broken down into what is known as quality models, which makes it possible to better specify them. Software quality models make it easier to understand the process of software engineering (Pressman, 2005).

Concepts for a Software Quality Model: Various software quality models are presented in the literature; some of them are product oriented (ISO 9126, McCall, FURSP, etc.) and other are process oriented (SPICE, CMM, PSP, etc.). However, the reference model considered in this research is ISO 9126-3 (ISO/IEC, 2001), which is a standard related to the internal quality of the software product. This is a general-purpose quality model and includes quality characteristics and metric examples (ISO/IEC, 2001).

ISO/IEC 9126-1 defines a set of quality characteristics and their respective sub-characteristics. ISO/IEC 9126-1 is used as a base for the construction of the three upper levels (characteristics, sub-characteristics and attributes) of the quality model. The quality model of software products described in ISO/IEC 9126-1 can be used to define the software product requirements, as well as a reference for the quality evaluation of a software product (ISO/IEC, 2001).

The characteristics of the ISO/IEC 9126 standard are: Functionality (suitability, accuracy, interoperability, compliance, security), Reliability (maturity, fault tolerance, recoverability), Usability (understandability, learnability, operability), Efficiency (time behaviour, resource utilization), Maintainability (analyzability, modifiability), and Portability (adaptability, instalability, suitability, replaceability).

Concepts for Quality Requirements: According to Kruchten (2003) software development process begins with a need expressed by the user or another stakeholder. This need is normally translated into one or more characteristics, which are part of the development vision. The characteristics expected from the system are translated into requirements representing the behaviour expected from the system in terms of both functionality and quality.

According to Whitten et al. (2004), software requirements are documented with a certain degree or thoroughness and through a set of specifications that help define the development scope. These specifications reflect not only the vision of those involved in the development, but also a set of constraints imposed by the business in form of rules limiting the own needs of the domain, which is clearly established in the Vision. The modern techniques used to specify functional requirements include the use-case models which are used to represent the behaviour of the system in response to the requests by each actor, taking into account that some supplementary characteristics of the system cannot be represented by means of models.

Therefore, requirement specifications are the basis for the definition of the design model or software architecture, as well as for other artefacts. Kruchten (2003) points out, then, that software architecture will be stable as long as it meets both functional as well as quality software requirements.

The last iteration in the methodology is applied to the concept of Evaluation Technique, identified in the first iteration.

Concepts for the Evaluation Technique: According to Bosch (2000), software architecture evaluation techniques help the architect measure some quality attributes.

Clements et al. (2002) classify the software architecture evaluation techniques into questioning techniques and measuring techniques. Questioning techniques include questionnaires, checklists and scenarios.

Clements et al. (2002) propose three kinds of scenarios: use-case, growth, and exploratory. Use-case scenarios reflect an interaction with the running system, foreseen by the users. Growth scenarios represent anticipated characteristics of future changes to the system. Exploratory scenarios are intended to expose the limits or conditions of the current design, exposing possibly implicit assumptions. Nowadays, techniques based on scenarios use two relevant evaluation instruments, namely: Utility Tree, proposed by Clements et al. (2002), and Profiles, proposed by Bosch (2000).

According to Clements et al. (2002), the Utility Tree is a tree-like scheme presenting the quality attributes of a software system. These attributes are refined into scenarios which specify, sufficiently in detail, each one's priority level. A Profile is a set of scenarios, generally with certain relative significance related to every one of them (Bosch, 2000).

Clements et al. (2002) establish that measuring techniques are used to answer specific questions about determined quality attributes. Measuring techniques use tools such as architectural description languages (ADL) and metrics, which are quantitative interpretations of particular observable measurements of the architecture. Metrics are used to measure the system complexity, determine how fault tolerance the modules are, etc.

When concepts in this section are considered together, a whole Conceptual Model representing the ontology is achieved (See Figure 1).

Once this approach is reached, an instantiation of the model is presented through a case study, as pointed out in the methodology. Since an approximation to the architectural evaluation of Collaboration Systems is a future research, a methodology specially developed for constructing this kind of software has been selected.

5 INSTANTIATION OF CONCEPTS BY EVALUATING COLLABORATION SYSTEMS

AMENITIES (Garrido, 2002) is a methodology used for the analysis and design of cooperative systems, inspired in the Unified Process for developing distributed systems. It is aimed at systematically addressing the analysis and design of the cooperative system facilitating further software development. The proposal comprises a specific set of models and phases to be followed. The general phases of the methodology are: 1) system analysis and requirement identification; 2) cooperative system modelling; 3) cooperative system analysis; 4) system design; and 5) software system development.

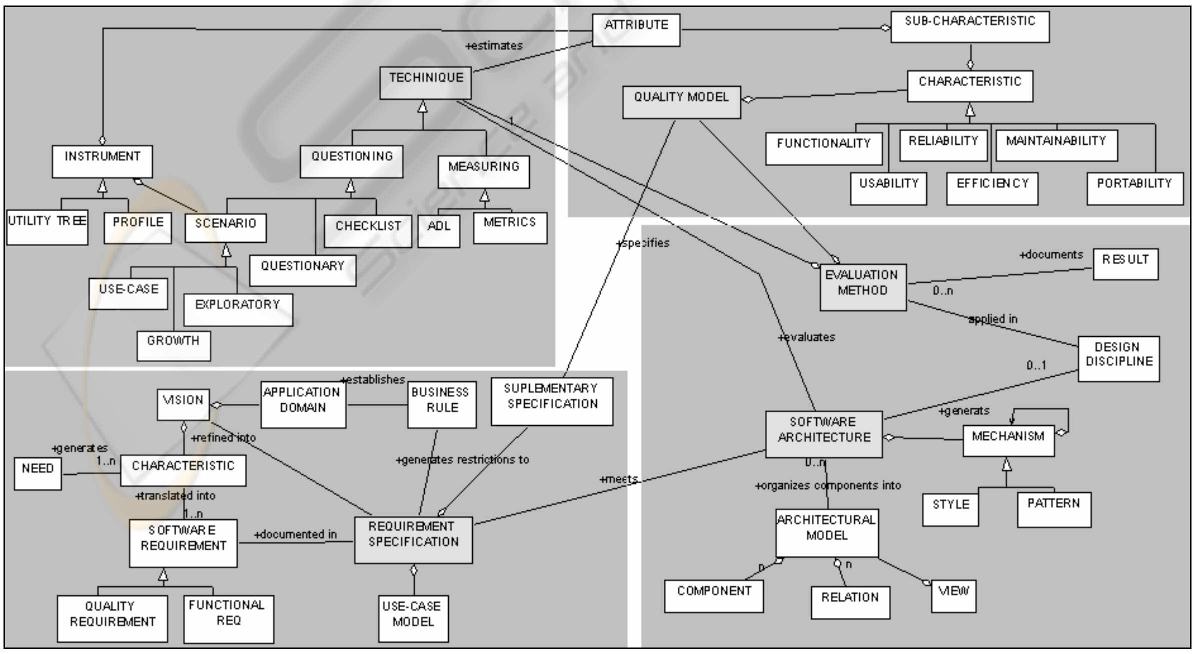

Figure 1: Integrated Conceptual Model for Architectural Evaluation.

A concepts instantiation is presented in order to evaluate the application of the previous ontology.

Domain: the software domain studied in this case includes a wide set of groupware supporting the cooperative and collaborative work, which impose diverse technical, socio-cultural, and organizational restrictions. In AMENITIES this concept is represented by the Requirement and Cooperative Model. They comprise the system requirements and the interactions between users. Cooperative Model include: Organizational, Interaction, Information, and Cognitive view.

Design: this discipline is present in AMENITIES as an activity that is related to Cooperative Model.

Software Architecture and Architectural Models: AMENITIES comprise a set of models developed since earliest phases, such as Use-Case Model, Formal Model, Cooperative Model, and Software Development Model. In this case, all of them could be considered for Evaluation. Software Development Model is specially related to software architecture.

Architectural views: AMENITIES includes a set of views in its Software Development Model, such as: Component, Functional, Dynamic, and Development view. Each view include packages or layer which represent architectural mechanisms, such as: Identification, Meta-information, Group Conscience, and Application Package.

Quality Requirement: this concept is represented in our case study by different non-functional requirements and socio-cultural restrictions. First ones include Efficiency, Portability, Maintainability and Evolution, Reliability, and -specially- Usability. Second ones is concerned with social and behaviour patterns in individual or groups. These requirements are not specified explicitly in AMENITIES; however, they could be collect and organized through a model, for example, ISO 9126 standard-compliant.

Functional Requirement: there are wide set of functional requirements for Collaboration Systems but we can identify that mostly they are related to cooperation, coordination, and communication. In addition, Interoperability and Security can be associated to this kind of software. Functional Requirements are represented in AMENITIES by the Use-Case, Cooperative, and Formal Models. These models also implicitly specified Business Rules, Needs and Characteristics, and establish the Vision of Collaboration System development.

Evaluation Techniques: this concept is not directly represented by any element in AMENITIES. It is incorporated in order to carry out the evaluation. In this sense, it is possible to apply a wide set of techniques depending on the evaluation objective and the available resources. Bellow, some techniques appropriate to evaluate Collaboration Systems Architectures, relating them to architectural models and quality attributes, as observed in our ontology.

– Objective: Validating the requirements identification based on Cooperative Model. Technique: Semantic analysis, traceability analysis, metrics. Quality Attributes: Functionality (structure, coordination and other collaborative process, accessibility of data-information and knowledge).
– Objective: Validating the Requirements identification based on Formal Model. Technique: Semantic analysis, simulation based on formal models, metrics, traceability analysis. Quality Attributes: Functionality (vivacity, feasibility, and persistence), Reliability (deadlocks), and Efficiency.
– Objectives: Estimating software quality characteristics based on Software Development Model. Selecting between different architectural decisions. Identifying sensitivity points of architecture, and identifying quality characteristics trade-offs. Technique: Scenarios (utility tree or profile), metrics, traceability analysis. Quality Attributes: Internal quality characteristics: Functionality, Reliability, Maintainability, Efficiency, Portability.

Once these concepts were instantiated, a dynamic representation proposing the evaluation incorporation in the process is reached. Figure 2 shows a dynamic view of the evaluation into AMENITIES process. Notice that needs and rules can be used in the whole process to validate requirements and architectural models. In addition, Use Case model collects functional requisites to be supported by software architecture. These elements are the main outputs of earliest phase, and they become inputs for architectural evaluation. At the same time, the outputs of design phase represent the bridge between analysis and construction phases.

Figure 2, also shows the traceability between different artefacts in the whole development process, which implies an attention point to architectural evaluation. It can be observed, a feedback loop provided by architectural evaluation that promotes decisions making about next activities or artefacts construction.

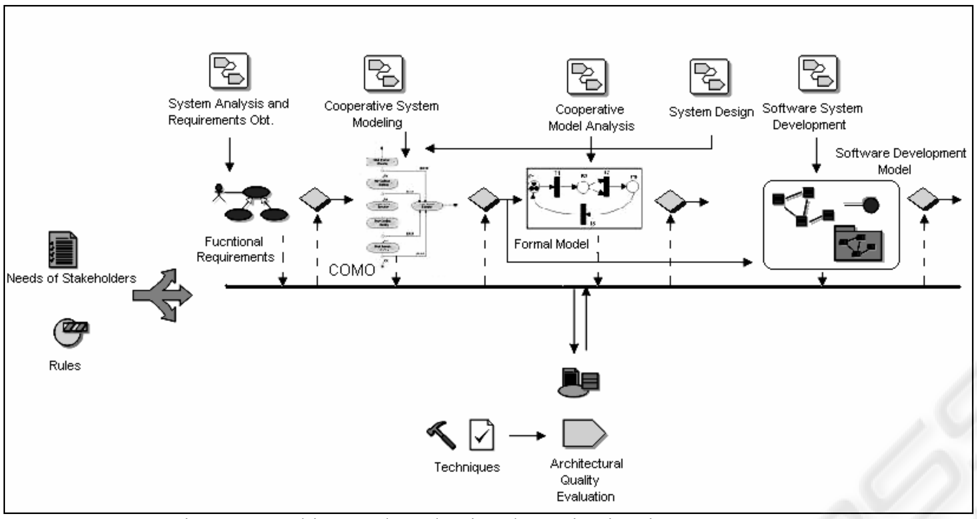
System Analysis and Requirements Obt.
Cooperative System Modeling
Cooperative Model Analysis
System Design
Software System Development
Software Development Model
Fucntional Requirements
COMO
Formal Model
Needs of Stakeholders
Rules
Techniques
Architectural Quality Evaluation

Figure 2: Architectural Evaluation dynamic view in AMENITIES.

6 CONCLUSIONS

As observed in our Conceptual Model, the Architectural Evaluation is a complex process that depends on a clear and complete requirement specification. It is also based on models and their relationship (traceability). In this sense, the previous Requirement Elicitation phase is fundamental to establish the right inputs. At the same time, architectural evaluation results will be considered in late evaluations.

The instantiation of concepts in AMENITIES allowed determining: (1) the necessity of a well-defined development process in terms of activities and artefacts, as well as their traceability; (2) the advantage of applying standard notation and formal model, as well as the use of multiple views in Software Architecture. Counting with these elements could implicitly guarantee some quality attributes, and facilitate the evaluation. Features researches will focus on the application of this proposal to an actual system development.

REFERENCES

Barbacci, M., Klein, M., Longstaff, T., & Weinstock, C. (1995) Quality Attributes. Carnegie Mellon University. Technical Report. Retrieved from: http://www.sei.cmu.edu/publications/documents/95.reports/95.tr.021.html

Bass, L., Clements, P. & Kazman, R. (2003) Software Architecture in Practice. Second Edition Addison Wesley Publising Co.

Bosch, J. (2000) Design & Use of Software Architectures. Addison-Wesley.

Buschmann, F.; Meuner, R.; Rohnert, H.; Sommerland, P.; Stal, M. (1996) A System of Patterns. John Willey & Sons.

Clements, P. Kazman, R y Klein M. (2002) Evaluating Software Architectures: Methods and Case Studies. The SEI Series in Software Engineering,

Garrido, J., Gea, M., Padilla, N., Cañas, J.., Waern, Y.(2002) AMENITIES:Modelado de Entornos Cooperativos. In: Aedo, I., Díaz, P., Fernández, C.(eds.):Actas de Interacción'02, Madrid, Spain , 97-104

Hofmeister, C.; Nord, R.; Soni D. (2000). Applied Software Architecture. Addison Wesley.

In, H., Kazman, R., y Olson, D. (2001). From Requirements Negotiation to Software Architectural Decisions. Software Engineering Institute, Carnegie Mellon University. Jasper and Uschold

ISO/IEC (2001) Software Engineering – Software quality – General overview, reference models and guide to Software Product Quality Requirements and Evaluation (SQuaRE). Report. JTC1/SC7/WG6

Kruchten, P. (2003). The Rational Unified Process. Reading, MA: Addison Wesley Longman, Inc.

Noy, N.. & McGuinness, D. (2001) Ontology Development 101: A Guide to Creating Your First Ontology. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 .

Pressman R. (2005) Software Engineering. A practical approach. (7a. ed.) Mc Graw Hill.

Shaw, M., and Garlan, D. (1996). Introduction to Software Architectures. New perspectives on an emerging discipline. Prentice Hall.

Sommerville, I. (2005) Software Engineering. Addison-Wesley. 6th Edition.

Whitten, J., Bentley, L., & Dittman, K. (2004) Systems Analysis and Design Methods. Sixth Edition. McGraw-Hill.