JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering **©JOT, 2002**

Vol. 1, no. 4, September-October 2002

Quality Models to Design Software Architecture[*]

Francisca Losavio, Central University of Venezuela, Caracas, Venezuela

Abstract

Quality requirements, captured in general as nonfunctional requirements in the early steps of software development, influence greatly the software system's architecture. However, also the system's core abstractions which are functional requirements, play an important role in the definition of the initial architecture. The importance of architectural design has grown rapidly in the last few years, since the need for reliable evolutionary systems and component-based development has increased. The goals of this work are to briefly discuss several architectural design approaches and to propose a systematic way of specifying the relevant quality attributes involved in the architectural design process. The evaluation of these attributes is the base of the architectural transformation process, allowing the incremental adaptation of the initial candidate architecture. This initial candidate, selected on some key functional requirements of the system, is adapted (transformed or refined) in the design process to fulfill the established quality goals. The SQUID (Software QUality In the Development process) approach, based on the standards ISO 9126-1 and ISO 14598-3, is used to define the quality model corresponding to the architecture and the development process model. These models will be constructed for Bosch architectural design method, which has been selected has a case study for offering very precise guidelines on the architecture transformation process. However, our approach could be easily integrated in other development process frameworks, like for example the Rational Unified Process, customizing the architectural construction process. We feel that the application of this approach is a step forward towards the systematization and improvement of the architectural design process, with built-in quality issues.

1 INTRODUCTION

The main goal of this work is to propose an improvement of the architectural design process by applying an ISO-based approach to quality, the SQUID approach [Bøegh et

[*]This work has been developed as a result of the European Community INCO SQUAD Project EP 962019 and the Consejo de Desarrollo Científico y Humanístico (CDCH) of the Central University of Venezuela, ARCAS project 03.13.4584.00

al.99] for quality specification, planning, control and evaluation, and the modeling of the software measurement supporting the activities to ensure software quality. In this work we use SQUID mostly for quality specification. In order to apply our approach, we will discuss several methods for architectural design and use one of them as a case study.

An important aspect of architectural design is that, on one hand quality requirements influence greatly the software architecture. On the other hand, the different requirements have to be "balanced" during the design process. Only recently the importance of an explicit design of software architecture has grown up considerably [Shaw,Garlan96], [Bosch00], [Jacobson et al.99] for the construction of reliable evolutionary systems. Modern applications involving distribution, portability, interoperability, component reusability and real-time approaches require an early definition of the system architecture in order to fulfill nonfunctional requirements, such as maintainability and reliability, which are crucial for the achievement of the overall functional purpose of the software system under construction. Nonfunctional requirements may appear during the functional requirements elicitation, but there are no explicit guidelines on how to capture them in standard object-oriented methods. Software architecture design should not be considered as an independent activity, but a step further in the development and evolutionary process of software products.

The Rational Unified Process (Inception and Elaboration phases)

General frameworks for software process development, such as the Unified Process (UP) [Jacobson et al.99] defined by Rational Software (RUP) [Krutchen00] presents general guidelines in this sense. The architecture is developed over iterations during the inception and the elaboration phases. The UP models can be modified in each iteration and the architecture is incrementally constructed. The architectural description contains views of the different models of the system, use case, analysis, design, implementation and deployment models. These different views of the models are organized in five main views [Krutchen00]. The central view is the *Use case view*, containing the key use cases constituting the core abstractions of the system, which are used in the Inception phase to select a candidate architecture. The *Use case Model* offers this view. The *Logical view* addressing the functional requirements, is an abstraction of the *Design Model* for identifying subsystems, classes, major packages. The *Process view* addresses concurrent aspects of the system at runtime, threads or processes, fault tolerance, response time and distribution issues. It is concerned with scalability. The *Design Model* is also used to describe these nonfunctional aspects. The *Implementation View* describes the components (source code) and other artifacts of the development environment. Finally the *Deployment view* describes how the various executable and other runtime components are mapped to the underlying platforms or composing physical nodes. The *Deployment Model* offers this view. The basic guidelines for constructing the architecture do not go deep into the details of how to select the core abstractions or how to specify the nonfunctional requirements. Since one of the main features of UP is to be use case centered, use cases are used to define the system's main components at a high level of abstraction, constituting the initial system's architecture, based on these main system's functionality or core abstractions. However, how to specify these nonfunctional requirements by means of the use cases, which are intended for capturing functional requirements, is not explained in details. A bad selection of the key use cases at this stage could compromise the whole software project. The Test Model is used to evaluate the architecture, but how to design the part of the Test model corresponding to check the architectural behavior against the established quality goals is left to the software engineer team customizing the UP.

Few traditional software development methods deal explicitly with quality architectural design. Few object-oriented methods propose to use explicitly the use case for nonfunctional requirements specification, such as Larman's method [Larman99] and Whitten's adaptation of Jacobson's approach [Whitten,Bentley01]. The approaches that will be discussed below are specific for architectural design, and could be included in general process frameworks.

The Bosch Method

Jan Bosch method [Bosch00] considers the design of software architectures taking account of the quality requirements from the early stages of development. The architectural design process, seen as an optimization problem, is viewed as a function taking as input the functional requirements specification and generating as output the architectural design. In the first step, a first version of the architecture is produced, not accounting for the quality requirements. Then, this design is evaluated with respect to the quality requirements. Each quality attribute is given an estimated value. These values are compared with the values of the quality requirement specification. If all the values are as good or better than required, the architectural design process is finished. Otherwise, a second step transforms the initial architecture, during which, quality values for some attribute improve. This design is again evaluated and the same process is repeated, if necessary, until all quality requirements are fulfilled or until the software engineer decides that there is no feasible solution. In this case the software architect needs to renegotiate the requirements with the customer. Each transformation (quality attribute-optimizing solution), generally improves one or some quality attributes, affecting others negatively. This method requires a formal or semi-formal specification of the architecture, for example using an ADL (Architecture Definition Language) [Shaw,Garlan96]. In this case a simulation of the runtime behavior of the architecture is possible.

The ABD method

ABD (Architecture Based Design) method [Bachmann et al.00] provides a structure to produce the *conceptual architecture* of a system. The conceptual architecture describes the system being designed in terms of the major design elements and the relationships among them [Hofmeister et al.00]. It represents the first design choices made during the development process, crucial to provide a basis for the achievement of the desired functionality. ABD determines the *architectural drivers* for a system. They are the combination of business, quality and functional requirements that influence the architecture. The ABD design activities begin as soon as the architectural drivers are set. This does not mean that the requirements elicitation, specification and analysis do not have to be completed, only that they can go in parallel with design activities. There are applications where it is not possible to determine in advance all the requirements, such as long-lived systems or product lines, hence the possibility of quickly start to design is important. ABD is founded on the following elements:

1. *Functional decomposition* using coupling and cohesion.
2. *Realization of quality* and business requirements through the choice of an architectural style.
3. Use of *software templates* to describe a software system of a particular type. It describes how all the elements of the type must interact with shared services and infrastructure. It describes responsibilities that pertain to all elements of that type.

For the second element, the realization of quality and business requirements, the ATAM (Attribute Tradeoffs Analysis Method) [Kazman et al.98] could be used. In this point, it is similar to the Bosch method, and it is proposed as a technique for understanding the tradeoffs points or dependencies among the attributes, inherent to architecture evaluation. It provides a way to evaluate software architecture's fitness with respect to multiple competing quality attributes. Since these attributes interact, the method helps to reason about architectural decisions that affect quality attribute interactions. ATAM follows a spiral model of design, postulating candidate architectures followed by analysis and risk mitigation, leading to refined architectures. The structure used for helping the reasoning is based on the ABAS (Attribute-Based Architectural Style) [Klein,Kazman99], which considers only one relevant attribute at a time. A quality model for the attribute is constructed. The reason claimed for using separate or concurrent analysis, is to allow individual attribute experts to bring their expertise independently. We differ from this point of view and we think that a global quality attribute model will facilitate a better picture of the quality model and corresponding measures of the attributes [Losavio et al.02]. This aspect will be discussed further in the subsequent section.

Discussion on the architectural design methods

It can be appreciated that Bosch and ABD method are similar in the early steps (ATAM nearly corresponds to step 2 of Bosch method [Bosch00], [Losavio,Chirinos99]. However, one of the major differences between these approaches is that Bosch method includes concrete guidelines on how to transform or refine the architecture in order to meet the quality requirements. ABD uses functional decomposition to arrive to the software templates. Quality (and also functional and business) requirements are used to verify the decisions taken during the decomposition Nevertheless neither ABD nor Bosch method propose a particular approach to construct a global quality model in order to arrive to the precise measurements of the quality attributes [Losavio et al.02]. The RUP approach to architectural design is embedded in a generic process framework that can be easily customized to any of Bosch or the ABD methods [Losavio,Chirinos00]. On the other hand, RUP and Bosch method coincide on the fact that the initial architecture must be selected on functionality basis, i.e. the core abstractions of the system. This initial rough architecture is then transformed in the architectural design process. Moreover, RUP considers the tradeoffs that are ATAM's main points, in the Test Model.

The following general principles of architectural design are proposed [Losavio et al.00] to synthesize the main activities of the architectural design methods discussed above.

1. Start with functional requirements
2. The nonfunctional (quality and business) requirements are captured
3. The design strategies can be based on reusable architectural styles or patterns
4. The architecture is designed by successive transformations

This work proposes an improvement of the architectural design process by applying an ISO-based approach to quality, the SQUID approach [Bøegh et al.99] for quality specification. We have had a nice experience in applying SQUID to an object-oriented method for interactive systems development [Losavio,Chirinos99], OOMGRIN (Object-Oriented Method for GRaphical user-INterface development) [Losavio,Matteo97]. After having applied directly ATAM's ABAS structure, we have proposed and used an extension of the structure, defining ISO 9126 [ISO98] quality models customized to interactive systems and to real-time applications [Losavio et al.00], respectively. This approach offers a better global picture of the quality attributes enriching the quality analysis step, allowing defining quantitatively the quality attributes. In this paper we will apply the SQUID approach to Bosch architectural design method, in order to specify the quality requirements, since it offers more precise guidelines for the architectural transformation process.

The structure of the paper, besides this introduction containing a survey on several known architectural design approaches, is the following: the second section describes briefly the SQUID approach. The third section discusses as a case study the use of the SQUID method for modeling Bosch's architectural design process. A quality model based on ISO 9126 is defined for software architecture, focused as a sub product of the design in the development process. Finally, in the conclusion the future work will be explored.

2 THE SQUID APPROACH

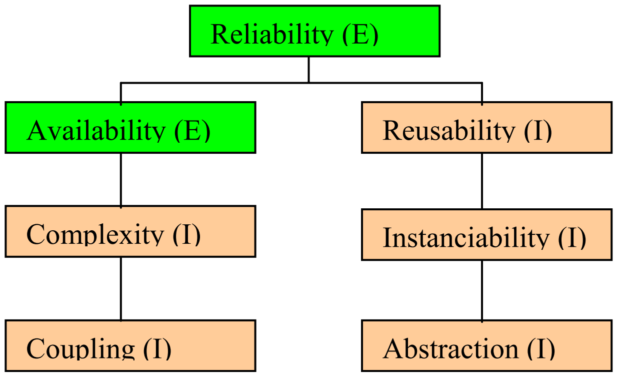
SQUID (Software Quality in the Development Process) [Bøegh et al.99] allows the specification, planning, evaluation and control of software quality through the software development process. Quality is defined as the operational behavior of a product required by its users. It offers a method and a tool supporting the method, both called SQUID. It uses external and internal quality measures defined in ISO 9126. External measures measure the quality characteristics in terms of the product's operational behavior. Internal measures can be collected during the product development process, to monitor and control the final product quality. Characteristic is a term used to define abstract properties that cannot be directly measured. Attributes instead, refer to directly measured properties. SQUID allows the organization to customize the quality model, based on the ISO 9126 and ISO 14598 [ISO98a], to the particular domain of the software applications to be constructed. The organization must define precisely the development process that has to be used in order to achieve the established quality requirements for the particular applications, contributing to the process improvement. In our case, this aspect is crucial, because we want to define a quality design process to build a solid baseline for the software architecture. SQUID supports the activities for quality specification, planning, control and evaluation and the modeling the software measurement that support these activities. This measurement approach is called configuration and is the SQUID step that will be used in this work.

Configuration step

1. Model the development process
   - The product portions or components are identified, establishing the requirements for each portion, until the attributes level.
   - Specification of the development model in terms of the project objects types: deliverables, development activities, review points
2. Specify the ISO 9126 quality model
   - Specification of the quality model in terms of the model elements: quality characteristics, external or internal quality sub-characteristics, quality attributes (directly measurable). The hierarchical link between characteristics and sub-characteristics is automatically established by the SQUID tool
3. Specify the measures
   - Specification of the units or data elements used to measure quantifiable attributes (not all the attributes are quantifiable)
   - Specification of the attribute values. These can be obtained by direct measurement, or defined as targets or estimates
4. Collate the models
   - Specification of the counting rules defining the condition under which a measure is obtained
   - Link the project objects quantified by a measurable attribute
   - Assignment of each internal attribute of the quality model to an appropriate project object type of the development model, associating a unit and counting rules to each attribute.

3 CASE STUDY: SQUID CONFIGURATION APPLIED TO BOSCH METHOD

In the architectural design process, the architecture will be considered the product to be constructed. Hence, the quality requirements for the architecture (product) must be established. Since the architecture depends greatly on the problem or application domain, the quality model can be established accordingly, customizing the ISO 9126 model. For example, a quality model for constructing soft real-time applications, like stock exchange monitoring, is shown in Figure 1 [Losavio et al.00]. Bosch method will be considered for the development process.



where I/E mean internal/external characteristics respectively
Fig. 1: Quality Model for Real-time Monitoring Systems

Definition of the objects of the development process

According to the SQUID approach, in order to define the conceptual model of software quality, customized to the development process, the development objects have to de identified. Measures can then be associated with the development objects and quality characteristics. As it has been pointed out, quality, according to ISO 8402 [ISO94], is expressed as the set of characteristics and properties of an item that affect its ability to satisfy established or implicit needs. Three types of objects are identified in the SQUID approach**:** *revision points* or milestones which are the control points for monitoring the process and *activities* that produce the *deliverables* which have to be inspected. An activity is a specific time period that has "begin" and "end" point in time; a revision point, instead is the end of an activity. These objects can be repeatable or not. We consider that each deliverable must be submitted to an inspection or a test, according to the kind of deliverable. The revision points, in our case correspond to the main steps of the development process and are illustrated in Table 1.

The model of Bosch development process

Revision points	Type	Description
1. Initial architecture	Non repeatable	Functionality-based architectural design. The archetypes are based on the designer's perception of the domain (domain analysis)
2. Evaluation of quality attributes	Non repeatable	Evaluate the potential of the architecture to reach the required levels for its quality requirements. The evaluation is with respect to a specific context or domain (Ex. GUI or real-time systems)
3. Transformation of the architecture	Repeatable	The estimated values of the quality attributes are compared to the requirements specification

Table 1. Revision points in the development model

Now, the activities for each revision point are specified in Table 2. The step of the process is used as the first number to identify the corresponding activity or deliverable or for each step. Notice that an activity can be a simple one or a complex one, requiring the application of a specific method or technique such as, for example the design of the scenarios. Actually, each activity could be decomposed into sub-activities. The activities shown here are the main ones. Notice also that the elaboration and delivery of inspection reports and tests are activities and deliverables that can be provided in each step, and they are omitted here in order to abridge this presentation.

Activities	Type	Description
1.1 Define the problem domain or context	Non repeatable	Define the characteristics of the domain according to the experience of the designer
1.2 Identify the archetypes	Repeatable	Identification of main components (core abstractions) of the system based on the functional requirements
1.3 Identify the structure	Non repeatable	Give the topology of the initial architecture
2.1Design the scenarios profile for a scenario-based evaluation	Repeatable	Design of the set of scenarios for the quality attributes (Ex. Change scenarios)
2.2 Define the Quality attribute profile	Non repeatable	Design of the quality model.
2.3 Establish the result of the evaluation	Non repeatable	Determine the potential of the candidate architecture to satisfy its quality requirements
3.1 Define the design decisions	Repeatable	Establish rules, constraints

Table 2. Activities identified in the development process

The deliverables produced by each activity are listed in Table 3.

Deliverables	Type	Description
1.1 Context of the system	Non repeatable	The problem domain category
1.2 Archetype	Repeatable	Core abstractions (Components)
1.3 Structure	Non repeatable	Topology
2.1 Scenario profile	Repeatable	A set of scenarios for the quality attributes (Ex. Change scenarios)
2.2 Quality attribute profile	Non repeatable	The quality model.
2.3 Result of the evaluation	Non repeatable	Potential of the candidate architecture to satisfy its quality requirements
3.1 Design decisions	Repeatable	Rules, constraints

Table 3. Deliverables in the development model

The quality model

The ISO 9126 standard for software quality measurement and the guidelines provided by ISO 14598 (Part 1, 2, 3) will be followed. The main quality characteristics are shown in Table 4.

Characteristic	Description
Functionality	The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions (what the software does to fulfil needs)
Reliability	The capability of the software product to maintain its level of performance under stated conditions for a stated period of time
Usability	The capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions (the effort needed for use)
Efficiency	The capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions
Maintainability	The capability of the software product to be modified. Modifications may include corrections, improvements or adaptations of the software to changes in the environment and in the requirements and functional specifications (the effort needed to be modified)
Portability	The capability of the software product to be transferred from one environment to another. The environment may include organizational, hardware or software environment

Table 4. Generic Quality Model, according to ISO 9126

It is clear that the ISO 9126 model is product-oriented, focusing the product external quality characteristics that must be accomplished when the product is in operation. However, the internal characteristics, which influence the external ones are taken into account. These internal characteristics arise during the development process and can be used to evaluate the architecture, which is a sub-product of the development process [Dromey86]. Moreover, SQUID is used here as a tool to model the development process according to the quality requirements defined in the quality model. At this point, two different approaches can be followed:

1. The generic model will be instantiated, like the example shown in Figure 1, according to the particular domain. External end internal characteristics will be defined accordingly, with the corresponding attributes. It corresponds to activity 2.2 (see Table 2). The SQUID configuration step could be completed for evaluating the particular architecture for the specific domain.

2. The generic model can be instantiated considering the general characteristic required in the architecture as a product itself. In this case, a general framework for determining the quality attributes for software architecture is obtained, taking into account of all the main characteristics of the ISO model. The sub-characteristics are refined into attributes, or measurable entities, which are adapted to software architecture [Losavio et al.03]. For example for *Reliability* we have:

   - **Maturity:** the capability of the software product to avoid failures, as a result of faults in the software. It is refined into an attribute Mean Time To Failure (MTTF) measured on the source code.
     At architectural level:
     1. The attribute is computed by the following metric:
        $$\Sigma_i \, Maturity\ (Component_i) + \Sigma_j \, Maturity\ (Connector_j).$$

     Notice that the Maturity attribute of the COTS components is known or should be.

   - **Fault tolerance:** the ability to maintain a specified level of performance in case of software fault or of infringement of its specified interface.
     At architectural level:
     1. It means to have a mechanism or software device. It may be a component or integrated into a component, for example exception handling or redundancy.
     2. It is refined into an attribute whose value is yes or not, depending on the presence or not of the mechanism or device.
     3. It can be refined into an attribute whose value is associated to the mechanism or device.

   - **Recoverability:** It is expressed by: 1. Capability to re-establish the level of performance. 2. Capability to recover the data. 3. Time and effort needed for it.
     At architectural level:
     1. It means the existence of a mechanism or software device, which may be a component or integrated into a component, to re-establish the level of performance or to recover the data, for example redundancy.
     2. If the mechanism exists, recoverability is refined into the attribute *performance* computed by metrics involving time and effort. It must be computed for each component holding the mechanism.

     Remarks:
     1. **availability** depends on the above three sub-characteristics of reliability. Even if this property is not directly specified in ISO 9126-1, it is defined as the capability of the software product to be in a state to perform a required function in a given period of time. It must be

considered for its importance in commonly used distributed and real-time application. It is like a fault tolerance attribute, measuring switching time.

2. **compliance** means in general to adhere to standards or regulations. Adheering to compliance would mean that the other sub-characteristics are not taken into account.

For further details on the attributes for the remaining sub-characteristics, the reader is referred to [Losavio et al.03].

According to SQUID, now the measures and model collation should be performed, after having defined the quality model. These activities are performed to accomplish activity 2.2 of Bosch's development model, to obtain the quality attribute profile as a deliverable. In this way, Bosch method is complemented by a more systematic and precise way to obtain all the measures, according to more quantitative reasoning. However, it should be noticed that the quality model is still greatly influenced by the architect expertise, according to a quite subjective approach.

4 CONCLUSION

In this work, after having reviewed the general aspects of several software architecture design methods, we have used a quantitative approach to complement the Bosch architectural design method for constructing the quality model to evaluate the quality attributes. Bosch method has been selected as a good candidate for this study, since it offers more precise guidelines on the architectural transformation process. The SQUID approach to control and monitor software quality in the development process has been used to define an ISO 9126-based quality model, considering software architecture as a sub-product of the design phase of the development process. An improved model of Bosch's development method has been defined in terms of the development objects, the revision points, activities and deliverables. In this sense, SQUID has been used for process improvement. We think that this work is a step forward towards the systematization of architectural design methods. We are now working on the precise definition of the measures and counting rules, in order to complete the SQUID configuration step in all its details. Another important aspect that will be explored in the near future, is the customization of the Unified Process, with respect to the inception and elaboration phases, to the ABD or Bosch methods.

This study is a step forward towards the definition of a method for architectural design that could be easily used as a customization of the RUP architectural design process or in any other general process framework. We feel that every method or process concerning modern applications involving, for example component-based development, should be provided with a solid architectural design method with built-in quality issues.

REFERENCES

[Bachmann et al.00] Bachmann F., Bass L., Chastek G., Donohoe P., Peruzzi F. “The Architecture Based Desing Method”. TR CMU/SEI-2000-TR-001 ESC-TR-2000-001, January 2000

[Bøegh et al.99] Bøegh J., DePanfilis S., Kitchenham B., Pasquini A. “A Method for Software Quality Planning, Control and Evaluation”. IEEE Software, 69-77, March/April 1999

[Bosch00] Bosch J. “Design and Use of Software Architecture”, Addison Wesley, Harlow, England, 2000

[Dromey96] Dromey, R., “Cornering the Chimera”, 33-43, IEEE Software Vol. 13, No.1, January 1996

[Hofmeister et al.00] Hofmeister, C., Nord R., Sony P. “Applied Software Architecture”, Reading MA, Addison Wesley, 2000

[ISO94] ISO/IEC 8402, “Quality Vocabulary”, 1994

[ISO98] ISO/IEC FCD 9126-1.2: Information Technology - Software Product Quality. Part 1: Quality Model, draft 1998.

[ISO98a] ISO/IEC 14598-3. Information Technology - Software Product Evaluation - Part 3: Process for Developers. Software Engineering. June, draft 1998

[Jacobson et al.99] Jacobson I, Booch G., Rumbaugh J. “The Unified Software Development Process”, Addison Wesley, Harlow, England, 1999

[Kazman et al.98] Kazman R., Klein M., Barbacci M., Longstaff T., Lipson H., Carriere J., “The Architecture Tradeoff Analysis Method”,CMU/SEI-98-TR-008, ESC-TR-98-008, July 1998

[Klein,Kazman99] Klein M., Kazman R., “Attribute-Based Architectural Styles”, CMU/SEI-99-TR-022, ESC-TR-99-022, October 1999.

[Krutchen00] Krutchen P. “The Rational Unified Process. An Introduction”, Second Edition, Addison-Wesley, Readings, Massachusetts, 2000

[Larman99] Larman C. “UML y Patrones”, Prentice Hall. Mexico 1999

[Losavio,Chirinos00]	Losavio F., Chirinos L. "Enfoques de Calidad en el Desarrollo de Software", L Convención Annual de ASOVAC, (363), Caracas, Noviembre 2000
[Losavio,Chirinos99]	Losavio F., Chirinos L., "Evaluación de la calidad en el desarrollo de sistemas interactivos", (92-108) Proceedings X CITS, Curitiba, Brazil, 17-21, May 1999.
[Losavio,Matteo97]	Losavio F, Matteo A., "A Method for User-Interface Development", Journal of Object- Oriented Programming, 10, 2, (22-27), Sept. 1997.
[Losavio et al.00]	Losavio F., Matteo A., Ordaz Jr. O., Levy N., Marcano-Kamenoff R. "Quality Characteristics to select an Architecture for Real-time Internet Applications", 4th Quality Week Europe, Brussels, November 2000
[Losavio et al.02]	Losavio F., Chirinos L., Perez M. "Attribute-Based Techniques to Evaluate Architectural Styles for Interactive Systems", to appear in Acta Científica Venezolana, Vol. 53, no. 2, 2002
[Losavio et al.03]	Losavio F., Chirinos L., Lévy N., Ramdane-Cherif A. "Quality Characteristics for Software Architecture", to appear in JOT 2003
[Shaw,Garlan96]	Show G., Garlan D. "Software Architecture. Perspectives on an Emerging Discipline", Prentice Hall, New Jersey, 1996
[Whitten,Bentley01]	Whitten J. L., Bentley L. D., Dittman K., C. "Systems Analysis and Design Methods", 5th. Edition, McGraw-Hill Irwin, 2001

About the author

Francisca Losavio received doctoral degrees in France, University of Paris-Sud, Orsay. She is head of the research Laboratory of Software Technology (LaTecS), Central University of Venezuela, Caracas, where she works at the Software Engineering post graduated studies of the Faculty of Science. Her research topics are software architecture and software quality. She can be reached at flosav@cantv.net